(2)

NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA

104pg

THESIS

AIR FORCE SPACE COMMAND SATELLITE ORBIT PREDICTOR USING PARALLEL VIRTUAL MACHINES

by

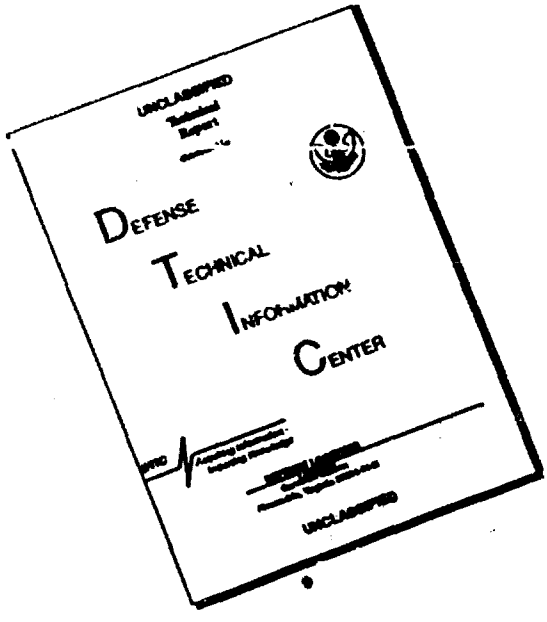
Susan K. (Matusiak) Brewer

December 1993

Thesis Advisors: Beny Neta
Don Danielson

Approved for public release; distribution is unlimited

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 1993	Master's Thesis

4. TITLE AND SUBTITLE

AIR FORCE SPACE COMMAND SATELLITE ORBIT PREDICTOR USING PARALLEL VIRTUAL MACHINES

5. FUNDING NUMBERS

6. AUTHOR(S)

BREWER (MATUSIAK), SUSAN K.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the policy or position of the Department of Defense or the United States Government

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Parallel computing is the wave of the future. As the need for computational power increases, one processor is no longer sufficient to achieve the speed necessary to solve today's complex problems. The Air Force Space Command (AFSPACECOM) tracks approximately 8000 satellites daily; the model used by the AFSPACECOM, SGP4 (Simplified General Perturbation Model Four), has been the operational model since 1976. This thesis contains a detailed discussion of the mathematical theory of the SGP4 model. The tracking of a satellite requires extensive calculations. The satellite can be tracked more efficiently with parallel processing techniques. The principles developed are applicable to a Naval ship tracking multiple incoming threats; the increase in the speed of processing incoming data would result in personnel being informed faster and thus allow more time for better decisions during combat. Three parallel algorithms applied to SGP4 for implementation on a Parallel Virtual Machine (PVM) are developed. PVM is a small software package that allows a network of computer workstations to appear as a single large distributed-memory parallel computer. This thesis contains a description of several algorithms for the implementation on PVM to track satellites, the optimal number of workstations, and methods of distributing data.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Parallel Virtual Machine - Satellite Orbit Prediction	104
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unclassified

Approved for public release; distribution is unlimited.

Air Force Space Command Satellite Orbit Predictor Using Parallel Virtual Machines

by

Susan Kay (Matusiak) Brewer
Lieutenant, United States Navy
BSMA, South Dakota School of Mines and Technology, 1987

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN APPLIED MATHEMATICS

from the

NAVAL POSTGRADUATE SCHOOL
December 1993

Author: Susan Kay (Matusiak) Brewer
Susan Kay (Matusiak) Brewer

Approved by: B. Neta
Beny Neta, Thesis Advisor

D.A. Danielson
Don Danielson, Thesis Advisor

Richard Franke
Richard Franke, Chairman
Department of Mathematics

ABSTRACT

Parallel computing is the wave of the future. As the need for computational power increases, one processor is no longer sufficient to achieve the speed necessary to solve today's complex problems.

The Air Force Space Command (AFSPACECOM) tracks approximately 8000 satellites daily; the model used by the AFSPACECOM, SGP4 (Simplified General Perturbation Model Four), has been the operational model since 1976. This thesis contains a detailed discussion of the mathematical theory of the SGP4 model.

The tracking of a satellite requires extensive calculations. The satellite can be tracked more efficiently with parallel processing techniques. The principles developed are applicable to a Naval ship tracking mulitple incoming threats; the increase in the speed of processing incoming data would result in personnel being informed faster and thus allow more time for better decisions during combat.

Three parallel algorithms applied to SGP4 for implementation on a Parallel Virtual Machine (PVM) are developed. PVM is a small software package that allows a network of computer workstations to appear as a single large distributed-memory parallel computer. This thesis contains a description of several algorithms for the implementation on PVM to track satellites, the optimal number of workstations, and methods of distributing data.

TABLE OF CONTENTS

ACKNOWLEDGMENT

To my husband, Terry, and daughters, Amanda and Sarah, for their support and love during the many hours required to accomplish this goal.

To my Father and Mother, Myron and Delores Matusiak, and my Grandmother, Albertina Greenwood, for their love and prayers to God to give me strength and knowledge. I am also thankful that they taught me the value of hard work and instilled in me a sense of confidence in my abilities.

To my thesis advisors, Beny Neta and Don Danielson, for their guidance and professionalism.

To my friends, Lieutenants Sara Ostrom and Traci Ford, for giving me outstanding answers to many technical questions.

Finally, to Al Geist and Vaidy Sunderam, the creators of PVM, for their cooperation in providing me with technical reports and advice necessary to complete this research.

I. INTRODUCTION

The goal of this thesis is to illustrate how a network of IPX Sunstations can be used as a parallel computer to solve a complex military requirement of tracking 8000 earth satellites daily. Parallel processing has already been used in Global Climate Modeling, Superconductivity, Seismic Imaging, and many other important applications in science today. Additionally, there are other important military applications where the use of parallel computing would be extremely advantageous. For example, today's Weapon Control Systems like AEGIS has enormous computational requirements to detect and destroy incoming threats. The use of separate computers located at individual enclaves versus a centrally located computer will reduce the vulnerability of a ship should it take a direct hit in the computer station. The necessary computing power will be continued by choosing unaffected stations; additionally, the increase in speed of processing incoming data would result in faster informed personnel and thus allow more time for better decisions during combat.

Parallel computing is the wave of the future. As the need for computational power increases daily, due to an increase in technological developments, one processor is no longer sufficient to achieve the speed in computations necessary to solve today's problems.

Two ways one can achieve greater computational efficiency with parallel processing are

1. Purchase a computer developed solely for parallel processing applications

or

2. Use existing workstations found in most companies today.

The first option requires the purchase of a computer like the INTEL iPSC/2 Hypercube multicomputer.

The INTEL iPSC/2 Hypercube at Naval Postgraduate school was purchased in 1987 for about $100,000.00; the Hypercube requires an additional $6000.00 per year to maintain, it is used solely for research projects.

The second option, the use of existing workstations, requires only that one be willing to utilize the power of idle workstation's CPU to achieve computational efficiency by dividing a complex problem into smaller more manageable data components.

The average computer user in the workplace today does not require 100 % of the CPU's power each hour of the day; additionally, at night the workstations remain idle until one logs in the next morning or after the weekend.

The utilization of thousands of existing processors to solve problems with enormous computational requirements will be common practice in the future. The price/performance advantage of this practice has not yet been fully realized; however, tomorrow's scientist will wonder how we achieved the advances in science and technology today with the use of serial processing alone.

Once one realizes that there is a storehouse of computer power ready to be distributed freely, the next step is to learn how to utilize this power. This thesis will illustrate how a network of workstations can be used to increase the speed at which satellites are tracked. This work will become increasingly more important as the number of objects tracked daily steadily increases and the number of calculations required skyrockets.

This is a continuation of the Parallel Processing Orbital Prediction work conducted at Naval Postgraduate School in the Mathematics Department orchestrated by Professors D.A. Danielson and B. Neta. In June 1992, Warren E. Phipps, Jr. developed several parallel

algorithms for the Naval Space Surveillance Center's analytic satellite motion model. The model is implemented in the FORTRAN subroutine PPT2. The algorithms were implemented on the INTEL iPSC/2 Hypercube (Phipps, 1992). In March 1993, Sara Ostrom studied the parallel computing potential of the Air Force Space Command analytic satellite motion model implemented on the INTEL iPSC/2 Hypercube (Ostrom, 1993). Currently, Leon Stone is implementing parallel algorithms for the Navy's Satellite model using Parallel Virtual Machines. This body of work is the result of the implementation of the Air Force Space Command's analytic satellite model, SGP4, using Parallel Virtual Machines.

Chapter II discusses the advantage of the Parallel Virtual Machine (PVM) in terms of cost, availability and fault tolerance factors. The history and components of PVM are discussed followed by a brief overview of a new extension to PVM called HeNCE. The chapter concludes with a short discussion of other parallel software packages available like Express, P4, and Linda. Chapter III describes the Air Force Space Command's analytical models SGP and SGP4 and describes, in detail, the theory behind the prediction of a satellite's position and velocity. Chapter IV describes three algorithms developed to study the parallelization of the satellite computer code; additionally, a comparison of the each algorithm's performance is analyzed in detail. The last chapter, Chapter V, contains conclusions and suggestions for further research.

II. PARALLEL VIRTUAL MACHINE

In this chapter, the advantages of using a Parallel Virtual Machine (PVM) in terms of cost, availability, and fault tolerance factors will be discussed. The history and components of PVM will be covered followed by a brief overview of a new extension to PVM called the Heterogeneous Network Computing Environment (HeNCE). Finally, other software packages like Express, P4, and Linda will be briefly described. This is a synthesis of papers written about the Parallel Virtual Machine (see Dongarra, Geist, Mancheck, and Sunderman ,1993).

Parallel Virtual Machine is a small software package (~ Mbyte of C source code) that allows a heterogeneous network of Unix-based computers to appear as a single large distributed-memory parallel computer. The PVM package is good for large-grain parallelism; that is, at least 100K bytes/node. The term *virtual machine* is used to designate a logical distributed-memory computer and *host* is used to designate one of the member computers.

The PVM software supplies the functions to automatically start up tasks on the virtual machine and allows the tasks to communicate and synchronize with each other. Note, a task is a unit of computation in PVM and is analogous to a UNIX process.

A problem can be solved in parallel by sending and receiving messages to accomplish multiple tasks. These message-passing constructs are common to most distributed-memory computers. By sending and receiving messages, multiple tasks of an

application can cooperate to solve a problem in parallel. The applications can be written in Fortran 77 or C.

PVM handles all message conversion that may be required if two computers use different data representations. PVM also includes many control and debugging features in its user-friendly interface. For instance, PVM ensures that error messages generated on a remote computer are displayed on the user's local screen.

PVM allows these application tasks to choose the architecture best suited to the solution. PVM also supports heterogeneity at the machine and network levels. At the machine level, computers with different data formats are supported as well as different serial, vector, and parallel architectures. At the network level, different network types can make up a Parallel Virtual Machine, for example, Ethernet, Fiber Distributed Data Interface (FDDI), token ring, etc.

Users of PVM can also configure their own parallel virtual machine, which can overlap with other users' virtual machines. Configuring a personal parallel virtual machine involves simply listing the names of the machines in a file that is read when PVM is started.

A. ADVANTAGES OF PVM

The first advantage of using PVM is a reduction in cost; it is and will continue to be costly to allocate large computing resources to each and every user. The beauty of using workstations for parallel processing is that a user of a workstation may not use the machine all the time, but may need more than what a single workstation can provide

when applications are to be run. Many scientists are discovering that their computational requirements are best served not by a single, monolithic machine but by a variety of distributed computing resources, linked by high-speed networks.

The second advantage in network-based concurrent computing is the ready availability of development and debugging tools. Typically, systems that operate on loosely coupled networks permit the direct use of editors, compilers, and debuggers that are available on individual machines; also, users are already familiar with the use and individual idiosyncrasies of each tool so that learning new skills is not necessary.

The third advantage is the potential fault tolerance of the network(s) and the processing elements. Most multiprocessors do not support such a facility; hardware or software failures in one of the processing elements often lead to a complete crash. Additionally, it is the opinion of the author, that for Naval applications using different workstations in different areas of a Naval ship can reduce vulnerability should the ship take a direct hit in a critical area. The computing power needed for a combat system like Aegis could be continued by choosing unaffected stations.

A study conducted by Eichelberger and Provencher (1993) explored using PVM to model a survivable AEGIS combat system for a CG47 Ticonderoga class AEGIS cruiser model. Present naval combat systems possess only manual reconfiguration and static rudimentary automatic reconfiguration schemes. The study concluded that there is a significant improvement in mission readiness when using a reconfigurable computer architecture.

B. HISTORY OF PVM

In the summer of 1989, at Oak Ridge National Laboratory (ORNL), the development of PVM software began and is now distributed freely in the interest of the advancement of science around the world. The driving force behind the initial popularity of PVM was the ability to get an excellent price performance ratio- better than any other computer system in the world. In general, a cluster of about 10 high performance workstations is potentially capable of solving a problem as fast as a supercomputer costing 20 times more; thus, PVM is rapidly becoming a *de facto* standard for distributed computing. How did all this begin? The following is a brief history of PVM's creation and it's creators:

Summer 1989:	Vaidy Sunderam designed and implemented the first version of Parallel Virtual Machine while visiting Oak Ridge National Laboratory.
Summer 1990:	Vaidy Sunderam and Al Geist refined the PVM software to develop a Fortran interface and several parallel applications; additionally, a graphical interface called XPVM was developed.
November 1990:	Al Geist developed a PVM version of large material science application code run on a network of IBM RS/6000's which won the 1990 Gordon Bell Prize for best price/performance ratio of any application in the world.

December 1990:	Sunderam and Geist entered their PVM research into the 1990 IBM Supercomputer competition and won first prize.
March 1991:	PVM 2.0 was developed by Bob Mancheck from PVM 1.0 - the earlier research version. PVM 2.0 was made publicly available through netlib@ornl.gov.
Summer 1991:	Sunderam, Geist, and Manchek began working on the design features of PVM 3.0 such as dynamic configuration and new routine names. Additionally, a digest for users to exchange information was set up at pvmlist@mathcs.emory.edu.
December 1991:	Beguelin began the development of a new software package called Xab, a monitor and debugger for PVM programs. This version can be obtained by contacting adam@cs.cmu.edu.
February 1992:	PVM 2.4 was released and HeNCE was made available through netlib@ornl.gov.
Summer 1992:	Geist and his student developed a package built on top of PVM 2.4 that dynamically load balances a users application.
February 1993 :	PVM 3.0 released.
April 1993:	PVM 3.1 released.
August 1993:	PVM 3.2 is released.To receive this software send email to netlib@ornl.gov with the message: **send index from pvm3** **or ftp from netlib2@cs.utk.edu directory pvm3.**

C. COMPONENTS OF PVM

The PVM system is actually composed of two parts , the daemon and a library of PVM interface routines.

The daemon is called *pvmd3 (*sometimes abbreviated *pvmd)* and resides on all the computers making up the virtual machine. Any user with a valid login can install this daemon on a machine. When the user desires to run a PVM application, he/she executes *pvmd3* on one of the computers which in turn starts up *pvmd3* on each of the computers making up the user-defined virtual machine. A PVM application can then be started from a Unix prompt on any of these computers.

The library of PVM interface routines contains routines for passing messages, spawning processes, coordinating tasks, and modifying the virtual machine. The user can call any of these routines and application programs must be linked with this library to use PVM.

D. APPLICATIONS

A variety of applications have been developed over the past few years using PVM. Below is a partial list of some of these applications:

* Material Science
* Global Climate Modeling
* Atmospheric, oceanic, and space studies
* Meteorological forecasting
* 3-D ground water modeling
* Superconductivity, molecular dynamics
* Monte Carlo CFD application

* 2-D and 3-D seismic imaging
* 3-D underground flow fields
* Particle simulation
* Distributed AVS flow visualization

As a result of this thesis , one can add **Orbital Prediction** to this list.

Application programs are composed of subtasks (or components) at a moderate level of granularity. The programs view the PVM system as a general and flexible parallel computing resource which may be accessed at three different modes:

1. **Transparent** - subtasks are automatically located at the most appropriate sites.
2. **Architecture-dependent** - subtasks specific for architecture execution are chosen by the user.
3. **Machine-specific** - subtasks are located on a particular machine to exploit particular strengths of individual machines.

During execution, multiple instances of each component or subtask may be initiated. Figure 2.1 on the next page illustrates a simplified architectural overview of the PVM system (see Geist and Sunderman , page 3, 1993) .

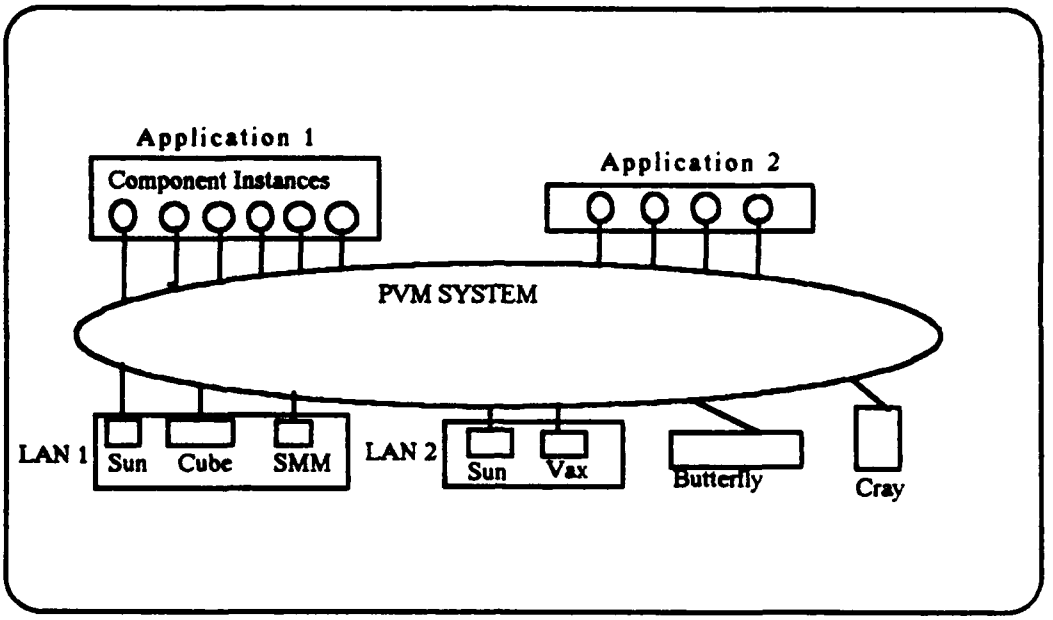


Figure 2.1 Simplified Architectural Overview of PVM

Application programs under PVM may possess arbitrary control and dependency structures; that is, at any point in the execution of a concurrent application, the processes in existence may have arbitrary relationships between each other and any process may communicate and/or synchronize with any other. Any specific control and dependency structure may be implemented under the PVM system by appropriate use of PVM constructs and host language control flow statements.

Multiprocessing on loosely coupled networks provides facilities that are normally not available on tightly coupled multiprocessors. For example, debugging support, fault tolerance, and profiling and monitoring to find hot-spots or load imbalances within an application.

The disadvantages associated with networked concurrent computing are generating and maintaining multiple object modules for different architectures, considerations of security into personal workstations, and other administrative functions. PVM supports two auxiliary components that provide some features to overcome these disadvantages. First, the HeNCE interface is a graphical based parallel programming paradigm. Second, PVM is undergoing extensions to make PVM work on MPP machines which it now does on several made by Intel, TMC, Cray, and Convex with KSR and Sequent underway (Geist, 1993).

E. HETEROGENEOUS NETWORK COMPUTING ENVIRONMENT (HeNCE)

HeNCE simplifies the writing of parallel programs and was developed with two goals in mind :

1. Make network computing accessible without the need for extensive training in parallel computing

   and

2. Make the resources best suited for a particular phase of the computation available to the users.

In HeNCE the programmer explicitly specifies parallelism of a computation by drawing graphs. The nodes in a graph represent user defined subroutines (written in either FORTRAN or C) and the edges indicate parallelism and control flow. HeNCE will automatically execute the subroutines in parallel (whenever possible) across a network of heterogeneous machines. HeNCE relies on the PVM system for process initialization

and communication. If one wishes to write explicit message passing parallel programs on a network of machines they should use the PVM system directly.

Once the graph is complete, HeNCE will automatically write the parallel program including all the communication and synchronization routines using PVM calls. HeNCE tools exist to assist the user in compiling this program for a heterogeneous environment.

HeNCE is composed of five integrated graphical tools. Below is a brief explanation of each tool:

1. **Compose** - use to specify the parallelism of an application by drawing a graph illustrating dependencies between procedures
2. **Configure** - use to specify a network of heterogeneous computers to be used as the PVM and defines a cost matrix between machines and procedures
3. **Build** - use to compile and install the procedures written by the compose tool
4. **Execute** - use to dynamically map procedures to machines for execution of the application and collect tracing information
5. **Trace** - use to read the trace information and display an animation of the execution, either in real time for debugging or later for performance analysis.

An initial version of HeNCE is available through the *netlib*. To obtain HeNCE send email to netlib@ornl.gov and next to subject one should type: **send index from hence**; any problems with HeNCE can be addressed to: **hence@msr.epm.ornl.gov**.

F. OTHER SOFTWARE PACKAGES

Various other software packages have been developed that enable scientists to write heterogeneous programs; these, as well as PVM, have evolved over the last several years, but none of them can be considered fully mature. It is an exciting time in parallel computing and there are many grand challenges for scientists to explore.

I would like to briefly discuss some of the other software packages, in order that the reader will be familiar with their names and features (see Dongarra, 1993). Examples of such other software packages include Express, P4, and Linda; however, it is important to note that these packages are by no means the only ones in existence. Each package is layered over the native operating systems, exploits distributed concurrent processing, and is flexible and general-purpose; all exhibit comparable performance. Their differences lie in their programming model, their implementation schemes, and their efficiency.

Express toolkit is a collection of tools that individually address various aspects of concurrent computation. The toolkit is developed and marketed commercially by ParaSoft Corporation, a company started by some members of the Caltech concurrent computation project. Express is based on beginning with a sequential version of an application and following a recommended development life cycle culminating in a

parallel version that is tuned for optimality. The core of the Express system is a set of libraries for communication, IO, and parallel graphics.

P4 is a library of macros and subroutines developed at Argonne National Laboratory for programming a variety of parallel machines. P4 supports both the shared-memory model and the distributed-memory model. In the process management mechanism in P4 there is a "master" process and "slave" processes, and multilevel hierarchies may be formed to implement what is termed a *cluster* model of computation. Shared Memory support via monitors is a distinguishing feature of P4; however, this feature is not distributed shared memory, but is a portable mechanism for shared address space programming in true shared memory multiprocessors. A set of macro extensions was developed at GMD (Gesellschaft für Mathematik und Datenverarbeitung in Schloss Birlinghoven, Gemany) called Parmacs. Parmacs provided Fortran interfaces and a variety of high-level abstractions dealing with global operations to the P4 system.

Linda is a concurrent programming model that has evolved from a Yale University research project. The primary concept in Linda is that of a "tuple-space", an abstraction via which cooperating processes communicate. The tuple-space concept is essentially an abstraction of distributed shared memory, with one important difference (tuple-spaces are associative), and several minor distinctions (destructive and non-destructive reads, and different coherency semantics are possible). Applications use the Linda model by embedding constructs that manipulate the tuple space. Recently, a new system technique has been proposed, at least nominally related to the Linda project.

This scheme, termed "Pirhana" proposes a proactive approach to concurrent computing where resources seize tasks from a well known location based on availability and suitability.

III. SGP AND SGP4

A. SIMPLIFIED GENERAL PERTURBATION MODEL(SGP)

The original model used by the Air Force Space Command to track satellites was the Simplified General Perturbation model (SGP). The model was simplified by the exclusion of perturbation effects caused by higher order terms in the Legendre expansion of the Earth's gravitational potential or other celestial bodies like the moon or the sun. The model also assumed the drag effect on mean motion as linear in time; this assumption dictated a quadratic variation of mean anomaly with time. The drag effect on eccentricity was modeled such that the perigee height remained constant (Hoots and Roehrich (1980), page 2).

These simplifications allowed an analytic solution to the equations of motion. Although the solutions are not as accurate as numerical techniques, they are computationally less expensive. Semi-analytic models increase the accuracy while decreasing the computational cost. See Dyar (1993) for comparison of various models in terms of accuracy and computer time required on a Sun Sparc 10.

Hilton and Kuhlman (1966) developed the analytical SGP model. SGP's gravitational submodel is a simplification of the work done by Kozai (1959) and Brouwer (1959). For a more detailed discussion of the SGP model see Hoots and Roehrich (1980) and Sara Ostrom (1993), pp. 10-20.

B. SIMPLIFIED GENERAL PERTURBATION MODEL FOUR (SGP4)

1. Overview

The second model, SGP4, was obtained by a simplification of a more extensive analytical theory developed by Lane and Cranford (1969) which uses the solution of Brouwer (1959) for its gravitational model and a power density function for its atmospheric model [Hoots and Roehrich (1980), p.2]. SGP4 had replaced SGP as the operational theory at the AFSPACECOM by 1976.

The SDP4 extension to SGP4 was developed to be valid for deep-space satellites. The deep-space equations were developed by Hujsak (1979). SDP4 models the effects of the moon and sun in addition to certain sectoral and tesseral Earth harmonics that become important for half-day and one-day period orbits.

The SGP4 and it's extension, SDP4, are both analytical models. They identify variations in terms of changes in the osculating elements with respect to time. The models are more accurate than the original SGP model due to two factors:

1. The inclusion of zonal harmonics through J_4; whereas, the SGP model only included zonal harmonics through J_3.
2. The inclusion of a drag force in the equations of motion versus the linear simplification of the SGP model.

The main program, DRIVER reads the input and calls either SGP4 or SDP4. If the satellite is "near-earth" (e.g., orbital period less than 225 minutes) then SGP4 is called; otherwise, the satellite is classified "deep-space" and DRIVER calls SDP4.

SGP4 and SDP4 receive input from the DRIVER and perform calculations necessary to return to the DRIVER the position and velocity vector in units of earth radii and minutes. The DRIVER performs a unit conversion to kilometers and seconds for printout.

SGP4 and SDP4 both call two functions, ACTAN and FMOD2P. ACTAN is passed the values of sine and cosine and returns the angle in radians in the range of 0 to 2π. FMOD2P is passed an angle in radians and returns the modulo by 2π of that angle.

Additionally, SDP4 calls the subroutine DEEP. The first time DEEP is called certain constants already calculated in SDP4 are passed through an entry called DPINT. All initialized quantities needed for deep-space prediction are calculated. At this time, it is also determined whether the orbit is sychronous or if the orbit experiences resonance effects. During initialization, the subroutine DEEP calls the function THETAG. The function THETAG obtains the location of Greenwich at epoch and converts epoch to minutes since 1950.

The next time SDP4 calls DEEP occurs during the secular update portion and is via the entry DPSEC. The secular update portion of SDP4 is where additional secular and long-period resonance effects are added to the values of the "mean" orbital elements.

The final access to DEEP occurs via DPPER where the appropriate deep-space lunar and solar periodics are added to the orbital elements.

2. Input Parameters

The SGP4 model uses the six orbital elements, a drag factor, and an epoch reference time to predict the satellite position and velocity vectors at a future time.

The six orbital elements are "mean" values obtained by removing periodic variations in a particular way. The elements are given below along with the name assigned to each in the SGP4 Fortran computer code:

VARIABLE NAME	SYMBOL IN THEORY	COMPUTER CODE
Mean Motion at Epoch	n_o	XNO
Eccentricity	e_o	EO
Inclination of Orbital Plane to the Equator	i_o	XINCO
Right Ascension of the Ascending Node	Ω_o	XNODEO
Argument of Perigee	ω_o	OMEGAO
Mean Anomoly at Epoch	M_o	XMAO

Table 3-1 Classical Orbital Elements

The following diagram will be useful throughout this discussion in visualizing the satellites orbit and the angles given in table 3-1 above:

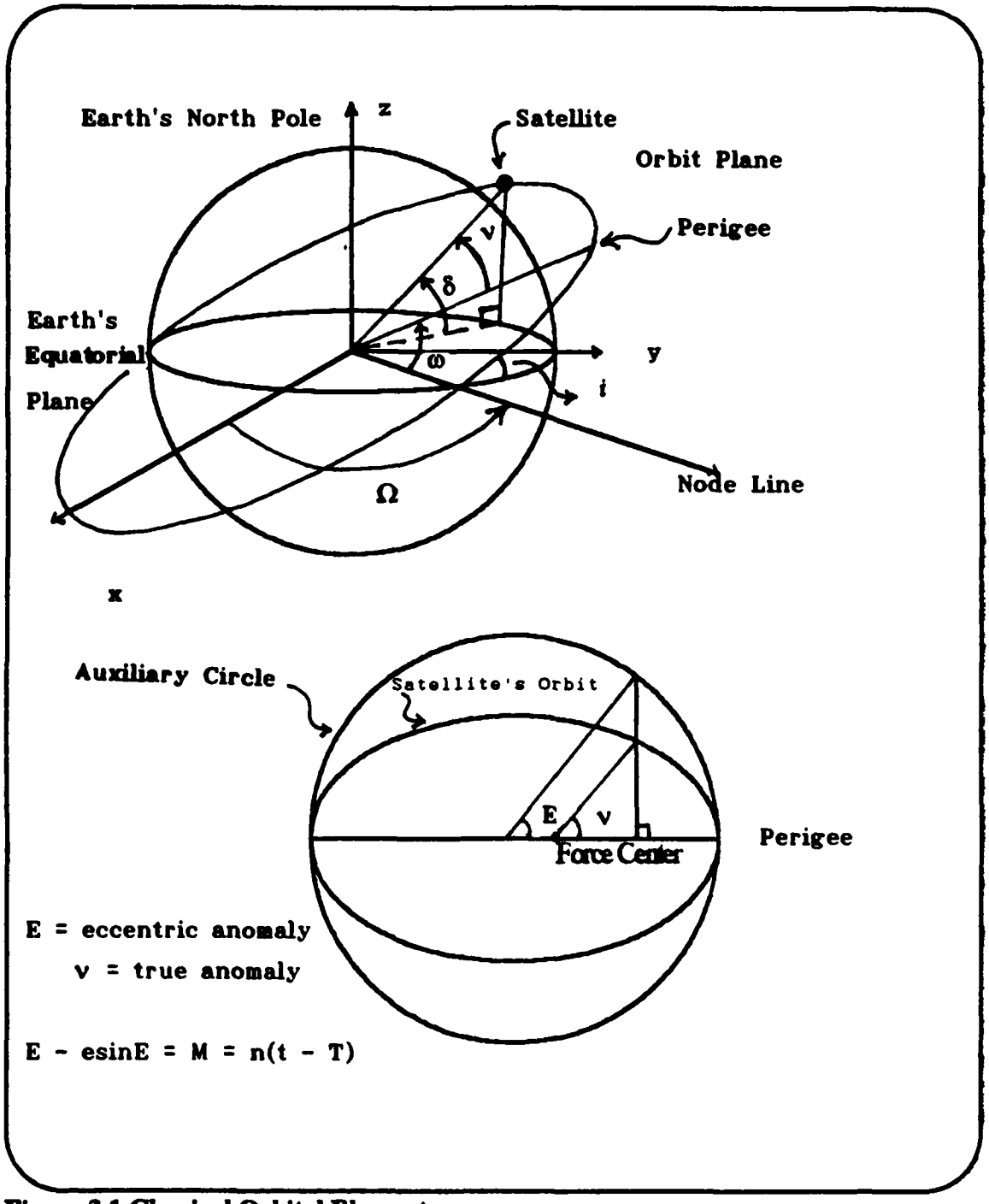


Figure 3.1 Classical Orbital Elements

3. PROGRAM SEQUENCE FLOW

The ten main steps to solve for position and velocity vectors are as follows:

1) Recover original mean motion and semimajor axis from the input elements.

2) If necessary, update the parameter for the SGP4 density function.

3) Calculate constants using appropriate values of the density function from step two above.

4) Account for the secular effects of atmospheric drag and gravitation.

5) Add the long periodic terms.

6) Solve Kepler's equation.

7) Calculate the preliminary quantities needed for short periodics.

8) Update the osculating quantities using the short periodics.

9) Calculate the unit orientation vectors.

10) Calculate the postion and velocity vectors.

The SDP4 model follows these same steps with the addition of several calls to the subroutine DEEP which was discussed earlier.

C. EQUATIONS

This section will describe the equations developed by Hoots and Roehrich (1980), pp. 14-37 . The ten main steps listed above will serve as the outline of the discussion. A strict parallel structure exists between the computer code and the equations.

1. Recover Original Mean Motion and Semimajor Axis

The input variable for mean motion (n_o) requires modification after which it is denoted by n_o''. This modification to n_o is accomplished as follows:

1) $n_o'' = \frac{n_o}{1+\delta_o}$ relationship of n_o'' to n_o

where

a. $\delta_o = \frac{3k_2(3\cos^2 i_o - 1)}{2a_o^2(1-e_o^2)^{3/2}}$

b. $k_2 = \frac{J_2 a_E^2}{2}$ J_2= the second gravitational zonal harmonic of the earth

a_E^2 = the equational radius of the earth squared

c. $a_o = a_1\left(1 - \frac{1}{3}\delta_1 - \delta_1^2 - \frac{134\delta_1^3}{81}\right)$

d. $\delta_1 = \frac{3k_2(\cos^2 i_o - 1)}{2a_1^2(1-e_o^2)^{3/2}}$

e. $a_1 = \left(\frac{k_e}{n_o}\right)^{2/3}$ where $k_e = \sqrt{GM}, G$ is Newton's universal gravitational constant and M is the mass of the Earth.

2) To recover the semimajor axis use $a_o'' = \frac{a_o}{1+\delta_o}$ where δ_o is the same as above.

2. Update The Parameter for the SGP4 Density Function

Two parameters, s and q_o, for the SGP4 density funcion may require adjustments. The scale height parameter constant used by SGP4 is

s = 1.01222928 earth radii (er); s changes depending on the height of the satellite at perigee. For perigees between 98 kilometers and 156 kilometers s is replaced by s^*, where $s^* = a_o''(1-e_o) - s + a_E$ with units of earth radii and where perigee height is calculated by perigee = $[a_o''(1-e_o) - a_E] \bullet R_E$ (kilometers) and R_E is the spherical earth radius.

For perigees below 98 kilometers, s is replaced by s^* where

$$s^* = \frac{20}{XKMPER} + a_E \qquad XKMPER = 6378.135 \text{ Kilometers/Earth radii}$$

It should be noted that if s is changed then a term $(q_o - s)^4$ is also replaced by $(qo - s^*)^4$.

From this point on, the double-prime notation will be dropped for the mean motion and the semimajor axis, as well as the * on s. It will be understood that these corrections have already been made when the symbols n_o, a_o and s are used.

3. Calculate Constants

a. The following constants are calculated for both SGP4 and SDP4:

$\theta = \cos i_o$

$\xi = \frac{1}{a_o - s}$

$\beta_o = (1 - e_o^2)^{1/2}$

$$\eta = a_o e_o \xi$$

$$C_1 = B^* \bullet C_2 \qquad B^* = \text{drag coefficient}$$

$$C_2 = (q_o - s)^4 \xi^4 n_o (1-\eta^2)^{-7/2} [a_o(1 + \tfrac{3}{2}\eta^2 + 4e_o\eta + e_o\eta^3) + \frac{3k_2\xi}{2(1-\eta^2)} \bullet \left(-\frac{1}{2} + \frac{3}{2}\theta^2\right)(8 + 24\eta^2 + 3\eta^4)]$$

$$C_4 = 2n_o(q_o - s)^4 \xi^4 a_o \beta_o (1-\eta^2)^{-7/2} \bullet \{[2\eta(1 - e_o\eta) + \frac{1}{2}e_o + \frac{1}{2}\eta^3] - \frac{2k_2\xi}{a_o(1-\eta^2)} \bullet [3(1-3\theta^2)(1 + \frac{3}{2}\eta^2 - 2e_o\eta - \frac{1}{2}e_o\eta^3) + \frac{3}{4}(1-\theta^2)(2\eta^2 - e_o\eta - e_o\eta^3)\cos 2\omega_o]\}$$

b. The following constants are calculated by SGP4 only for perigees above 220 kilometers:

$$C_3 = \frac{(q_o - s)^4 \xi^4 A_{3,0} n_o a_E \sin i_o}{k_2 e_o} \qquad \text{where } A_{3,0} = -J_3 a_E^3$$

$$C_5 = 2(qo - s)^4 \xi^4 a_o \beta_o^2 (1-\eta^2)^{-7/2} [1 + \tfrac{11}{4}\eta(\eta + e_o) + e_o\eta^3]$$

$$D_2 = 4a_o\xi C_1^2$$

$$D_3 = \frac{4}{3}a_o\xi^2(17a_o + s)C_1^3$$

$$D_4 = \frac{2}{3}a_o\xi^3(221a_o + 31s)C_1^4$$

4. Secular Effects of Atmospheric Drag and Gravitation

M_o , ω_o , and Ω_o are updated as follows:

a. First, M_{DF} , ω_{DF} , and Ω_{DF} are calulated:

1) $M_{DF} = M_o + \dot{M}\,\Delta t$

2) $\omega_{DF} = \omega_o + \dot{\omega}\,\Delta t$

3) $\Omega_{DF} = \Omega_o + \dot{\Omega}\,\Delta t$

where $\Delta t = t - t_o$ = time since epoch and

$$\dot{M} = \left[1 + \frac{3k_2(-1+3\theta^2)}{2a_o^2\beta_o^3} + \frac{3k_2^2(13-78\theta^2+137\theta^4)}{16a_o^4\beta_o^7}\right] \bullet n_o$$

$$\dot{\omega} = \left[\frac{-3k_2(1-5\theta^2)}{2a_o^2\beta_o^4} + \frac{3k_2^2(7-114\theta^2+395\theta^{4)})}{16a_o^4\beta_o^8} + \frac{5k_4(3-36\theta^2+49\theta^4)}{4a_o^4\beta_o^8}\right] \bullet n_o$$

$$\dot{\Omega} = \left[\frac{-3k_2\theta}{2a_o^2\beta_o^4} + \frac{3k_2^2(4\theta-19\theta^3)}{2a_o^4\beta_o^8} + \frac{5k_4\theta(3-7\theta^2)}{2a_o^4\beta_o^8}\right] \bullet n_o$$

Recall that , $k_2 = \frac{1}{2}J_2a_E^2$ J_2 = the second gravitational zonal harmonic of the Earth

and $k_4 = \frac{-3}{8}J_4a_E^4$ J_4 = the fourth gravitational zonal harmonic of the Earth

Note : this is the point in SDP4 where the DEEP initialization for deep-space calculations is entered through DPINT discussed earlier.

b. Then M_p , ω , and Ω are calculated by

1) $M_p = M_{DF} + \delta\omega + \delta M$

2) $\omega = \omega_{DF} - \delta\omega - \delta M$

3) $\Omega = \Omega_{DF} - \dfrac{21 n_o k_2 \theta C_1 \Delta t^2}{2 a_o^2 \beta_o^2}$

If perigee is less than 220 kilometers

$\delta\omega = \delta M = 0$

otherwise,

$\delta\omega = B^* C_3 (\cos\omega_o)\Delta t$

$\delta M = \dfrac{-2}{3}(q_o - s)^4 B^* \xi^4 \dfrac{a_E}{e_o \eta}[(1 + \eta\cos M_{DF})^3 - (1 + \eta\cos M_o)^3]$

Note: At this point SDP4 calls the secular portion of DEEP via DPSEC to add the deep-space secular effects and long-period resonance effects to the six orbital elements.

c. Next, e , a , and the mean longitude, L , are updated as follows:

1) $e = e_o - B^* C_4 \Delta t - B^* C_5 (\sin M_p - \sin M_o)$

2) $a = a_o[1 - C_1\Delta t - D_2\Delta t^2 - D_3\Delta t^3 - D_4\Delta t^4]^2$

3) $L = M_p + \omega + \Omega + n_o \bullet [\frac{3}{2}C_1\Delta t^2 + (D_2 + 2C_1^2)\Delta t^3$
$+\frac{1}{4}(3D_3 + 12C_1D_2 + 10C_1^3)\Delta t^4$
$+\frac{1}{5}(3D_4 + 12C_1D_3 + 6D_2^2 + 30C_1^2D_2 + 15C_1^4)\Delta t^5]$

If the perigee height is less than 220 kilometers then a and L equations are truncated after the C_1 term and the equation for e is truncated after the C_4 term.

d. The last step in this section is to calculate β and n :

1) $\beta = \sqrt{1 - e^2}$

2) $n = \frac{k_e}{a^{3/2}}$

Note: At this point SDP4 calls the periodics section of DEEP via DPPER to add the deep-space lunar and solar periodics to the orbital elements.

5. Add The Long Periodic Terms

The addition of long-periodic zonal effects are accomplished by the following:

a. $a_{XN} = e\cos\omega$

b. $a_{YN} = e\sin\omega + a_{YNL}$ where, $a_{YNL} = \frac{A_{3,0}\sin i_o}{4k_2a\beta^2}$

a_{XN} and a_{YN} are the horizontal and vertical components, respectively, of the eccentricity vector with respect to the line of nodes vector. The following figure illustrates the geometry of the components:

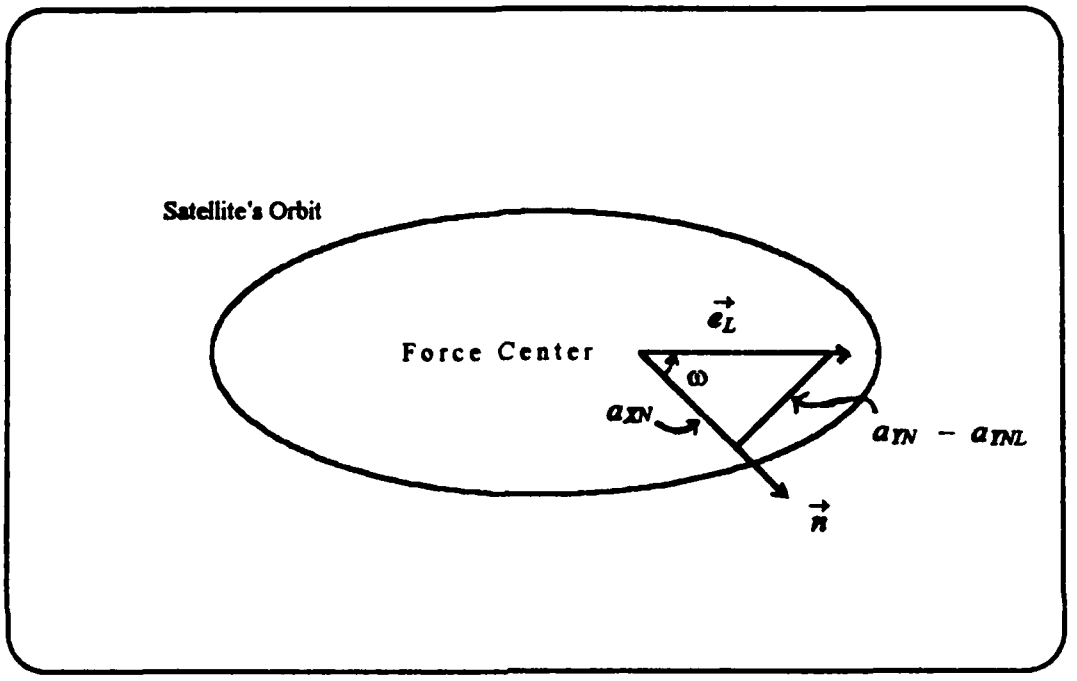


Figure 3-2 Geometry of Eccentricity Vector and Node Vector

The mean longitude is then calculated by:

$$L_T = L + L_L$$

where, $$L_L = \frac{A_{3,0}\sin i_o}{8k_2 a\beta^2} a_{XN}\left(\frac{3+5\theta}{1+\theta}\right).$$

Recall that L was calculated in the previous section.

6. Solve Kepler's Equation

Solve Kepler's equation by a method of successive approximations.

Let $U = L_T - \Omega$

and $U = (E + \omega)_1$ the first term in the iteration of the sum of the eccentric anomoly and the resulting argument of perigee. Thus,

$$U = U_o + \Delta U$$

for successive iterations, that is

$$(E + \omega)_{i+1} = (E + \omega)_i + \Delta(E + \omega)_i$$

Let $EPW = E + \omega$ then

$$\Delta(EPW)_i = \frac{U - a_{IN}\cos(EPW)_i + a_{XN}\sin(EPW)_i - (EPW)}{-a_{IN}\sin(EPW)_i - a_{XN}\cos(EPW)_i + 1}$$

Continue iterations until $|\Delta(EPW)_i| < 1.0^{-6}$ then set $E + \omega = (E + \omega)_1$.

7. Short Periodic Preliminary Calculations

The following equations are the preliminary calculations, the results are added in section eight to obtain the osculating quantities:

a. $e\cos E = a_{XN}\cos(EPW) + a_{IN}\sin(EPW)$

$$e\sin E = a_{XN}\sin(EPW) - a_{IN}\cos(EPW)$$

b. $e_L = \left(a_{XN}^2 + a_{YN}^2\right)^{1/2}$

c. $p_L = a(1 - e_L^2)$

d. $r = a(1 - e\cos E)$

e. $\dot{r} = k_e \frac{\sqrt{a}}{r} e \sin E$

f. $r\dot{f} = \frac{k_e \sqrt{p_L}}{r}$

g. $Temp3 = \frac{1}{1 + \sqrt{1 - e_L^2}}$

h. $\cos u = \frac{a}{r}[\cos(EPW) - a_{XN} + a_{YN}(e \sin E) \bullet Temp3]$

i. $\sin u = \frac{a}{r}[\sin(EPW) - a_{YN} - a_{XN}(e \sin E) \bullet Temp3]$

j. $u = \arctan\left(\frac{\sin u}{\cos u}\right)$

k. $\Delta r = \frac{k_2}{2p_L}(1 - \theta^2)\cos 2u$

l. $\Delta u = -\frac{k_2}{4p_L^2}(7\theta^2 - 1)\sin 2u$

m. $\Delta\Omega = \frac{3k_2\theta}{2p_L^2}\sin 2u$

n. $$\Delta i = \frac{3k_2\theta}{2p_L^2}\sin i_o \cos 2u$$

o. $$\Delta \dot{r} = -\frac{k_2 n}{p_L}(1-\theta^2)\sin 2u$$

p. $$\Delta r\dot{f} = \frac{k_2 n}{p_L}\left[(1-\theta^2)\cos 2u - \frac{3}{2}(1-3\theta^2)\right]$$

8. Update The Osculating Quantities

Now, the short periodic preliminary results are added to obtain the osculating quantities:

a. $$r_K = r\left[1 - \frac{3}{2}\frac{k_2\sqrt{1-e_L^2}}{p_L^2}(3\theta^2 - 1)\right] + \Delta r$$

b. $$u_K = u + \Delta u$$

c. $$\Omega_K = \Omega + \Delta\Omega$$

d. $$i_K = i + \Delta i$$

e. $$\dot{r}_K = \dot{r} + \Delta\dot{r}$$

f. $$r\dot{f}_K = r\dot{f} + \Delta r\dot{f}$$

9. Calculate Unit Orientation Vectors

The osculating angles found above are utilized to find the unit orientation vectors as follows:

$$\overline{M} = \begin{bmatrix} M_X \\ M_Y \\ M_Z \end{bmatrix} = \begin{bmatrix} -\sin\Omega_K \cos i_K \\ \cos\Omega_K \cos i_K \\ \sin i_K \end{bmatrix}$$

$$\overline{N} = \begin{bmatrix} N_X \\ N_Y \\ N_Z \end{bmatrix} = \begin{bmatrix} \cos\Omega_K \\ \sin\Omega_K \\ 0 \end{bmatrix}$$

then $\overline{U} = \overline{M}\sin u_K + \overline{N}\cos u_K$

and $\overline{V} = \overline{M}\cos u_K - \overline{N}\sin u_K$

10. Calculate The Postion And Velocity Vectors

Finally, the position and velocity vectors are calculated as follows:

$$\overline{r} = r_K \overline{U}$$

$$\dot{\overline{r}} = \dot{r}_K \overline{U} + \left(r\dot{f}\right)_k \overline{V}$$

This results in the position and velocity in units of earth radii and minutes. The postion and velocity vectors are then passed to the DRIVER at which time the unit conversion to kilometers and seconds is accomplished.

IV. PARALLELIZATION OF SGP4 USING PVM

A. OVERVIEW

The goals of this chapter are two-fold:

1. Explain how the Air Force Space Command's satellite code was parallelized using the Parallel Virtual Machine and
2. Compare various algorithms in terms of total time, communication overhead, speedup, and efficiency.
   a. Speedup (S_p) is calculated as follows:

$$Sp = \frac{T_1}{T_p}$$

where

T_1 = Endtoend Time on a Single Processor
T_p = Endtoend Time on p Processors

Note: Endtoend Time will be the term used to denote the total time to execute the program not including the time to read the input file.

b. Efficiency is calculated by:

$$E = \frac{S_p}{p}$$

where

S_p = Speedup for p processors
p = Number of processors

Three algorithms were developed to study the performance of the parallelization of the satellite code. The algorithms were based upon previous work completed by Ford and Carvahlo (1993).

Data was collected for each algorithm; each execution time is the result of an average of ten recorded run times.

Analysis was performed on each algorithm's results by comparing each model's performance and the use of four, eight, and sixteen nodes to execute the tasks.

It is important to note that with the use of an open network of computers there is undoubtedly going to be fluctuating machine and network loads. Multiple users and other competing PVM tasks cause the machine and network loads to change dynamically; thus, in order to have sufficient balancing, great care was taken to collect data at times where the load on the system was relatively constant. However, due to the fluctuation of open networks, the reproduction of the exact data results would be impossible.

In addition to the system load discussed above, one needs to consider Load Balancing. Load Balancing refers to the degree to which all nodes are working to solve the problem at hand. There are generally three types of Load Balancing according to Geist (1993):

1. **Static Load Balancing**

   The problem is divided into separate tasks which are assigned to the processors only once. The number or size of each task can be varied to utilize different computational powers of machines.

2. **Dynamic Load Balancing by Pool of Tasks**

   This is usually used with a Master and Slave scheme, the master continues to deal tasks to idle slaves until the task queue is empty. This results in the faster processors receiving more tasks.

3. **Dynamic Load Balancing by Coordination**

   Typically used by Single Program Multiple Data Stream (SPMD) where each processor receives a single set of instructions, receives and manipulates data, and redistributes its work at fixed times.

The second type, Dynamic Load Balancing by a Pool of Tasks, where a Master and Slave scheme exists was utilized in this research.

The Master/Slave approach is currently a popular distributed programming scheme. The Master starts all the Slave tasks and coordinates their work and input/output. All three algorithms developed use a Master/Slave approach.

Two other distributed programming schemes are the "hostless" Single Program Multiple Data (SPMD) and the Functional schemes (Geist, 1993). The "hostless" SPMD uses the same program executed on different pieces of the problem; whereas, the Functional scheme consists of several programs each one performs a different function in the application.

B. INPUT DATA

Approximately 8000 satellites are tracked by the Air Force Space Command (AFSPACECOM) in Colorado Springs daily; thus, a file consisting of 8000 satellite entries was created. Note that the same near-earth record and deep-space record was copied to generate the 8000 input records.

Each entry or input record consists of twenty-two individual numerical values. Table 4-1 on the following page illustrates a typical input record used.

Note that the input record used by AFSPACECOM consists of seventeen individual numerical values (see Hoots and Roehrich,1980, p.91) . Table 4-2 on page 39 illustrates a typical AFSPACECOM record.

There is a direct correspondence between the first 17 values of the input record used in this research and the first 16 values of the AFSPACECOM record. The seventeenth entry in the AFSPACECOM record is the epoch revolutions that have been recorded since the object was first launched. Note that this information is not used to calculate the position and velocity vectors of the satellite.

The entries 18 -22 in Table 4-1 simulate the number of calls made either to SGP4 or SDP4 per input record as will be explained later.

	Name	Explanation	Example
1	Cardno	2 card format	1
2	Satellite number	Satellite ID	88888
3	YR	Year	93
4	RDAY	Day	275.98708465
5	XNDOT	Derivative of mean motion	0.01431103
6	XN2DT	2nd derivative of mean motion	0.00000000
7	IE	Exponent of XN2DT	0
8	BTERM	Drag term	0.14311
9	IE2	Exponent of BTERM	-1
10	EPHTYP	Ephemeris type	0
11	ICRDNO2	Card number 2	2
12	XINCO	Inclination	46.7916
13	XNODEO	Right ascension	230.4354
14	EO	Eccentricity	0.7318036
15	OMEGAO	Argument of perigee	47.4722
16	XMAO	Mean Anomaly	10.4117
17	XNO	Mean motion	2.28537848
18	IYR	Start year	93
19	SRDAY	Start day	276.98708465
20	JYR	Stop year	93
21	SPDAY	Stop day	277.98708465
22	DELTA	Time step in minutes	60

Table 4-1 Example of Input Record

	Name	Explanation	Example
1	Cardno	2 card format	1
2	Satellite number	Satellite ID	00603U
3	International designator	Year/Launch No./ Piece	193-022B
4	Epoch time	Year and day-1st 2 digits are the year the others are the day	93162.71380248
5	$\frac{\dot{n}_o}{2}$ or B	Mean motion derivative(rev/day^2) or B (m^2/kg)	0.00073094
6	mean motion dot dot	Mean motion 2nd derivative/6	0
7	BSTAR	Drag term (er^{-1}) : the -3 is the exponent	45562-3
8	Ephtype	Denotes model : 2 is for SGP4	2
9	Element No.	Element number	864
10	Satellite number	Satellite number of card 2	00603U
11	i_o	Inclination	89.8623
12	Ω_o	Right ascension	245.9276
13	e_o	Eccentricity	.0006273
14	ω_o	Argument of perigee	337.4473
15	M_o	Mean anomaly	22.6464
16	n_o	Mean motion (rev/day)	15.03410461
17	Epoch rev	Epoch revolutions	59663

Table 4-2 Typical Input values for AFSPACECOM

The entry number 17 in Table 4-1 and entry number 16 in Table 4-2, the mean motion (XNO), determines whether or not the satellite is a deep-space object. SGP4 propagates data for near-earth satellites which require more frequent tracking due to the atmospheric drag factor and SDP4 propagates data for the deep-space satellites.

In order for an object to be classified as deep-space the period must be greater than 225 minutes. The period is calculated by

$$T = \frac{2\pi}{XNO}\left(\frac{days}{rev}\right).$$

For a period greater than 225 minutes XNO must be less than 6.4 since:

$$T = \frac{2\pi}{XNO}\left(\frac{day}{Rev}\right)\left(\frac{24\,hour}{day}\right)\left(\frac{60\ \text{min}}{hour}\right)\left(\frac{Rev}{2\pi}\right) \quad > \quad 225 \text{ minutes}$$

Rearrange and solve for XNO :

$$XNO \; < \; \frac{1440\ \text{min}}{225\ \text{min}}$$

That is,

$$XNO \; < \; 6.4\frac{Rev}{day}$$

Thus, the example in Table 4-1 illustrates a deep-space satellite and the example in Table 4-2 illustrates a near-earth satellite.

Out of the 8000 satellite tracked approximately 85 % are near-earth and 15 % are deep-space; therefore, 6800 of the 8000 input records (consisting of 22 elements each) were near-earth and the remaining 1200 records were deep-space.

The requirement for more frequent tracking of near-earth satellites was simulated by requiring 72 calls to the SGP4 subroutine per input record, resulting in 72 output records generated per input record. If the satellite was deep-space the SDP4 subroutine was called 24 times per input record, resulting in 24 output records generated per input record. 72 and 24 was choosen to parallel the work done by Ostrom (1993). The output record consisted of the time since the last propagation, three components of the position vector, and three components of the velocity vector for a total of 7 output data elements per output record.

To illustrate how this was accomplished, consider the input record in Table 4-1. The difference between the start year and day is one day or 1440 minutes. The time step of 60 minutes/call (over a period of 1440 minutes) resulted in 24 calls to the SDP4 subroutine.

C. ALGORITHMS

1. Overview

Three algorithms were considered in order to maximize load balancing and minimize communication overhead. All three algorithms used PVM to simulate a 2D torus topology. A 2D torus is like a 2D mesh with the addition of communication links between the nodes located at the "edge" of the mesh.

2. METHODS

a. Sequential

The Sequential program was developed to be the most efficient obtainable, in order to ensure the record of speedup values would not be misleading.

(1) Sequential Algorithm

READ DATA FILE
REPEAT
CALL PROPAGATION SUBROUTINE
UNTIL all input records have been converted to position and velocity vectors
COLLECT timing statistics

The sequential program can be found in Appendix A.

b. Parallel

In the following discussion the term "node" will denote one Unix-based workstation in a given network;specifically, one SUN microsystem SPARC station IPX.

In order to maximize the load balancing, a dynamic load balancing method by a pool of tasks was utilized. One node was designated the "Master" while the other nodes became the "Slaves". One of the slave nodes was designated as a collecting node. A separate collecting node is an advantage over having the master collect, since collection will begin before distribution is complete. This is also similar to the configuration used by Phipps (1992) and Ostrom (1993) in their work on parallel orbit prediction on the INTEL Hypercube.

When four nodes were utilized one node acted as the master and dealt tasks to two working nodes to complete. The remaining node acted as the collector by collecting the results from the working nodes and returning the results to the master. The research conducted by Ford and Carvalho (1993) concluded that a separate collecting node is a definite advantage over having the master collect, since collection can begin even before the distribution is complete.

In a similar fashion, when eight nodes were utilized there was a total of 6 working nodes and when sixteen nodes were utilized there was a total of fourteen working nodes.

3. Parallel Algorithms

a. Answer Back Method (ABM)

The first approach was to minimize the time a worker spent idle waiting for more data. The requirement was that the slave notify the master when it had completed it's tasks and was ready for more data. This would result in the fastest workers processing the most data. The algorithm for the Master Program is as follows:

```
READ entire satellite catalog input file
ENROLL in PVM and spawn n + 1 slaves
DESIGNATE 1 collector and n workers
REPEAT
PACK m sets of satellite input records
SEND data to worker
UNTIL each worker has m sets each
REPEAT
  PACK m sets of satellite input records
  WAIT until worker sends ready signal
  SEND data to worker
UNTIL all complete sets of m have been sent
```

```
REPEAT
  PACK any leftover satellite input records
  WAIT until worker sends ready signal
  SEND data to worker
UNTIL 8000 input records have been sent
SEND stop signal to workers
WAIT for program complete signal from collector
GATHER and compute timing statistics from slaves.
```

The algorithm for the Answer Back slave program is as follows:

```
INITIALIZATION
IF I am the collecting node
  REPEAT
  WAIT for one set of results
  STORE results
  UNTIL all results have been collected from the workers
 SEND program complete signal to master
ELSE
     I'm  a working node
REPEAT
  WAIT for data packet from master
  REPEAT
    UNPACK data
    CALL propagation subroutine
    PACK results
    SEND results to the collector
  UNTIL no more input records in the packet
 SEND ready for more data signal to the master
UNTIL master sends stop signal
END IF.
```

The Answer Back program can be found in Appendix A.

b. Successive Deal Methods

The second and third algorithms were developed to decrease the communication time between the master and slaves. The input records were dealt to the workers in sets

m at a time. After giving each worker an initial set, the master continued to deal input records until all 8000 records had been sent.

The successive deal methods are basically the same, the difference lies in the way the input data is dealt to each worker.

In the second algorithm (Successive Deal Model I), to study the result of sending larger data packets, each worker is dealt an input data set consisting of m records with 22 elements each. Next, 1/(2*p) of the remaining records are dealt to each worker. Finally, 1/p of the remaining records is dealt to each worker. Note that if any records are leftover as a result of the integer division, the leftovers are sent last. For example, if

n = number of data records
m = number of records sent simultaneously
p = number of working processors or nodes
s = sets of m records to be distributed.

and we let, n = 8000
m = 15
p = 2

Then, the number of sets to be distributed is $s = \frac{8000\,\text{records}}{15\,\text{records/set}} = 533\,\text{sets of } 15$ with 5 input records leftover. Now, a set is sent to each worker leaving a total of 531 sets left to be distributed. Next, 1/(2*p) records are dealt to each worker; that is,

$$\left(\frac{1}{2*p}\right) \bullet (531\,sets) = 132 \text{ sets are given to each worker.}$$

Thus, the number of sets left to be distributed is

$$s = 531 - (2*132) = 267 \text{ sets.}$$

Next, 1/p records are dealt to each worker; that is , (1/2)*267 sets = 133 sets are distributed leaving 1 set leftover. Finally, the leftovers are sent to a worker and all the input records have been distributed.

In the third algorithm, the Successive Deal Model II, the master deals out one set consisting of m input records to each worker. Then, the master continues to deal out data sets until all the records have been distributed. For example, using the variables defined above, let

n = 8000
m = 15
p = 2

then,

$$s = \frac{8000\,records}{15\,records/set} = 533\,sets \;+\; 5\,records\;leftover.$$

First, one set is given to each worker, resulting in 531 sets left. Then, the sets would be distributed, one at a time, first to one worker and then to the other worker. Last, the leftover records are sent.

(1) Successive Deal Method I (SDI) Algorithm

Master Algorithm

```
READ entire satellite catalog input file
ENROLL in PVM and spawn n + 1 slaves
DESIGNATE 1 collector and n workers
REPEAT
  PACK one set of m input records
  SEND data to worker
UNTIL each worker has one set
REPEAT
  PACK 1/(2*p) records
  SEND data to worker
UNTIL each worker has a packet
```

REPEAT
  PACK remaining sets
  SEND data to worker
UNTIL each worker has a equal packet
REPEAT
  PACK leftovers
  SEND leftovers
UNTIL all input records have been sent
SEND stop signal to workers
WAIT for program complete signal from collector
GATHER and compute timing statistics from slaves.

Slave Algorithm:

INITIALIZATION
IF I am the collecting node
  REPEAT
  WAIT for one set of results
  STORE results
  UNTIL all results have been collected from the workers
 SEND program complete signal to master
ELSE
    I'm a working node
REPEAT
  WAIT for data packet from master
  REPEAT
    UNPACK data
    CALL propagation subroutine
    PACK results
    SEND results to the collector
  UNTIL no more input records in the packet
UNTIL master sends stop signal
END IF

(2) Successive Deal Model II (SDII) Algorithm

Master Algorithm:

```
READ entire satellite catalog input file
ENROLL in PVM and spawn n + 1 slaves
DESIGNATE 1 collector and n workers
REPEAT
  PACK one set of m input records
  SEND one set to each worker
UNTIL each worker has one set
REPEAT
  PACK m sets of input records
  SEND data to worker
UNTIL all m sets have been distributed
REPEAT
  PACK remaining input records
  SEND data to worker
UNTIL all input records have been distributed
SEND stop signal to workers
WAIT for program complete signal from collector
GATHER and compute timing statistics from slaves.
```

Slave Algorithm:

```
INITIALIZATION
IF I am the collecting node
  REPEAT
  WAIT for one set of results
  STORE results
  UNTIL all results have been collected from the workers
 SEND program complete signal to master
ELSE
     I'm a working node
REPEAT
  WAIT for data packet from master
  REPEAT
    UNPACK data
    CALL propagation subroutine
    PACK results
    SEND results to the collector
  UNTIL no more input records in the packet
UNTIL master sends stop signal
END IF.
```

For the source code of the algorithms discussed above see Appendix A. The programs developed were written in C. The SGP4 code is written in FORTRAN. The C framework using a PVM architecture calling a FORTRAN satellite propagation subroutine was successful.

D. PROGRAM OVERVIEW

1. Sequential

The sequential version was executed 10 times and the total run times were averaged. This was done four times and the four average values were averaged resulting in a sequential time T_1, which is used in the calculation of speedup.

The total time for the program to execute did not include the initial time to read the entire input catalog because this was done one time only at the beginning of each program. From this point on the total time to execute the program , excluding readtime will be called endtoend time. The sequential average endtoend time was used in the calculation of speedup which will be discussed in the Parallel section below.

2. Parallel

In each program discussed under the Parallel Algorithm section above, time clocks were inserted at various locations in order to measure the time to read the entire input catalog, the endtoend time, the worker's communication time, and the worker's calculation time.

The number of satellite input records (consisting of the 22 input values) sent simultaneously to each worker was chosen to be either 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, or 55. This was based upon previous work done by Ford and Carvalho (1993).

The number of nodes utilized was 4, 8, or 16. To configure the personal parallel virtual machine, a list of names of the Unix-based machines used was listed in a file called hostfile. When PVM was started by the command pvmd3 hostfile & , the hostfile was automatically read and the machines were ready to act as nodes in a parallel application.

The machine from which the application was started acted as the master and the slave nodes were spawned by first specifying the number of nodes desired (num_nodes) and then executing the statement

num = pvm_spawn(SLAVENAME, (char**) 0, 0, "", num_nodes, tids).

The selection of 4, 8, or 16 nodes was based upon previous work done by Ostrom (1993) in the parallelization of the SGP4 code using the Naval Postgraduate School INTEL iPSC/2 Hypercube. This is a Multiple Instruction stream, Multiple Data stream (MIMD) multicomputer. It consists of a system resource manager called the host, and eight individual processors, referred to as nodes.

Data for each set of choices discussed above was collected for ten iterations of the entire program and these results were averaged.

a. Analysis

For endtoend time, percent worker communication, speedup, and efficiency, two comparisons were analyzed to measure the performance of each algorithm:

(1) For a given algorithm, the performance of four, eight, and sixteen nodes utilized was compared and

(2) For a given number of nodes, the three algorithm's performance was compared.

For both cases above the number of satellite input records sent simultaneously was either 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, or 55.

It is important to note that for all cases, the same input record was utilized; thus, for all three models the number of calls made to SGP4 and SDP4 was the same.

E. RESULTS

1. Read Time

The time to read the data file (consisting of 8000 records) varied from approximately 39 seconds to 1100 seconds. Thus, the readtime was extremely dependent of the load on the system at the time the data file was read. This was in contrast to the results found by Ford and Carvahlo (1993); the number of input records used in their research was 630 and the read time was approximately 5 seconds for each execution.

2. Endtoend Time

The endtoend time is the most important time considered because it is a reflection of the total performance of each algorithm designed.

a. Method Comparison

For 4 and 8 nodes, the optimal performance was achieved by the Answer Back Method (ABM). For 16 nodes , with the exception of sending 15, 50, or 55 records at a time the ABM was superior. That is, when sending 5, 10, 20, 25, 30, 35, 40, and 45 records simultaneously, the ABM produced the fastest times.

From this point on in this analysis, when a given algorithm is superior the majority of the cases (as shown above) the term "in general" will be used. For the case above, one would say "When 16 nodes were utilized, in general, the Answer Back Method (ABM) was the best." The following graphs illustrate these results:

(1) Using four nodes the Answer Back Method was the fastest:

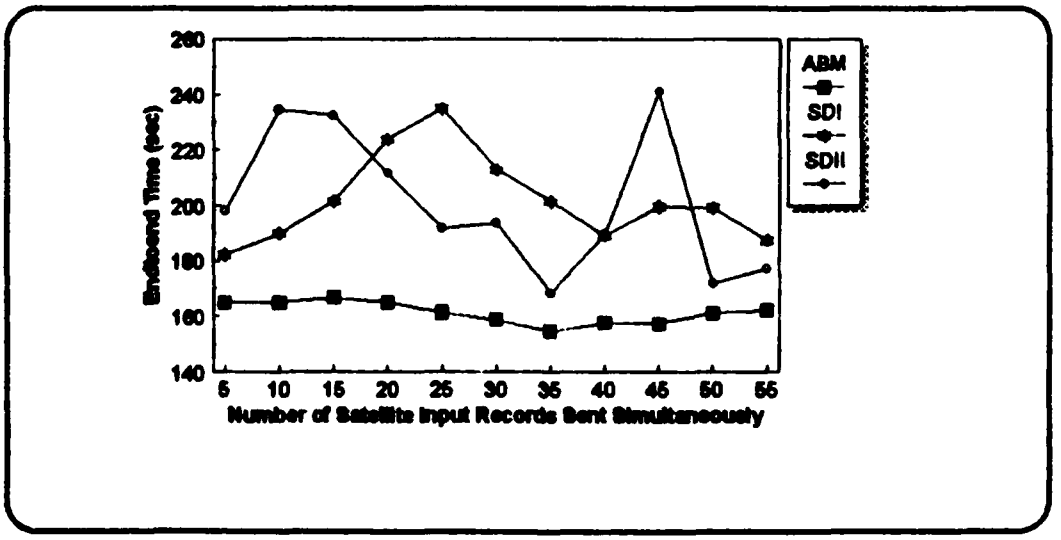


Figure 4.1 Four Node Comparison of Models

(2) Using eight nodes the Answer Back Method was the fastest.

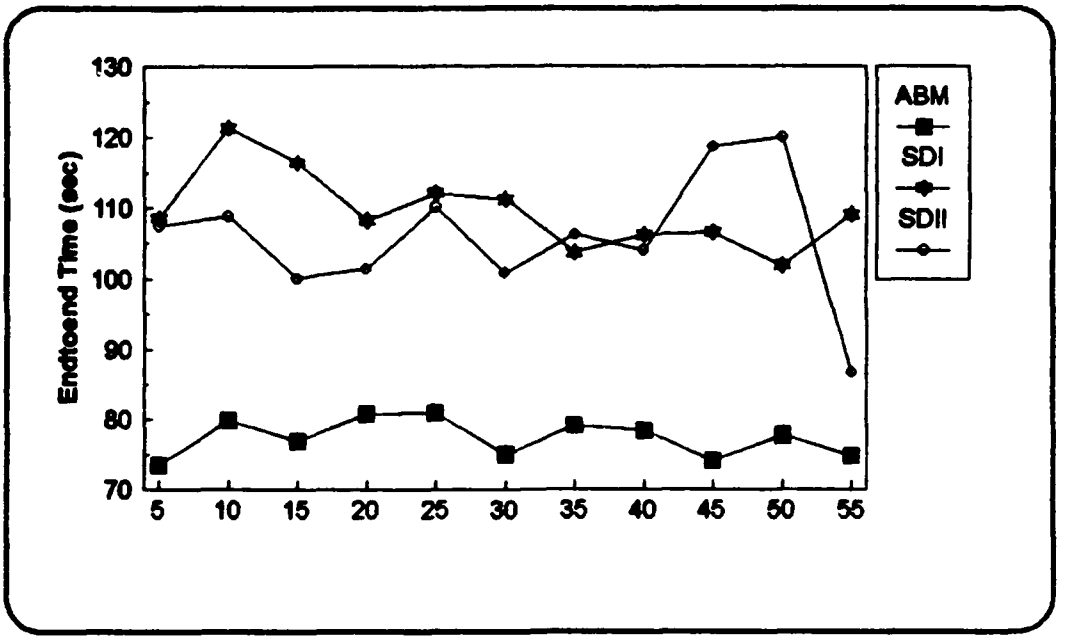


Figure 4.2 Eight Node Comparison of Models

(3) Using sixteen nodes, in general, the Answer Back Method was fastest.

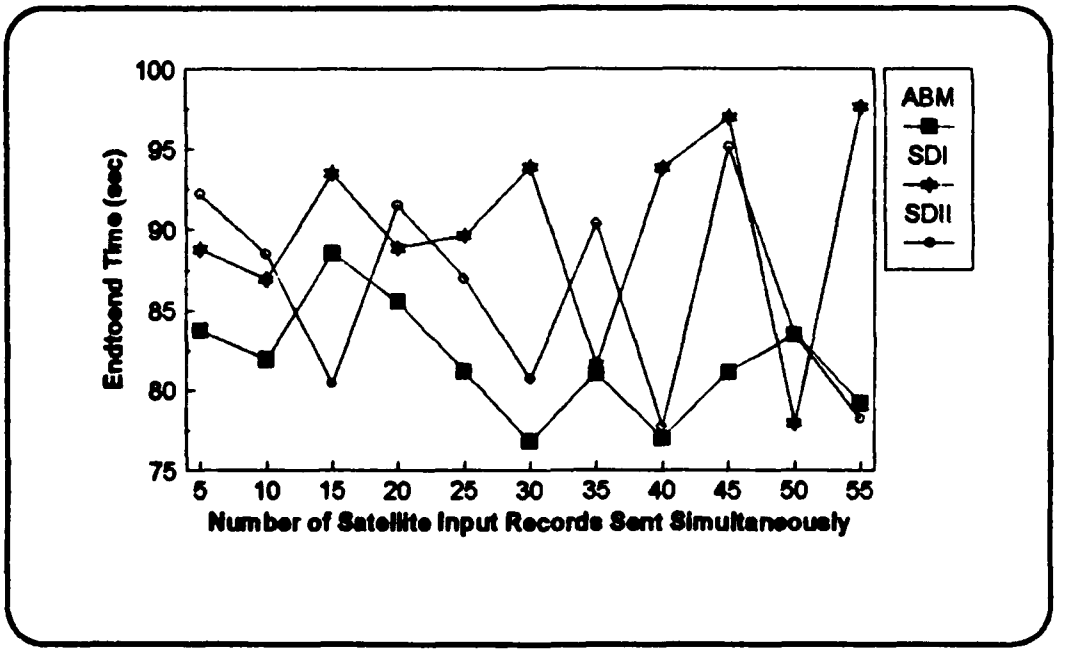


Figure 4.3 Sixteen Node Comparison of Models

b. Node Comparison

For the analysis comparing the performance of various choices of nodes for a given algorithm the following conclusions can be made:

(1) For the Answer Back Method, a choice of eight nodes was the best; closely followed by sixteen nodes.

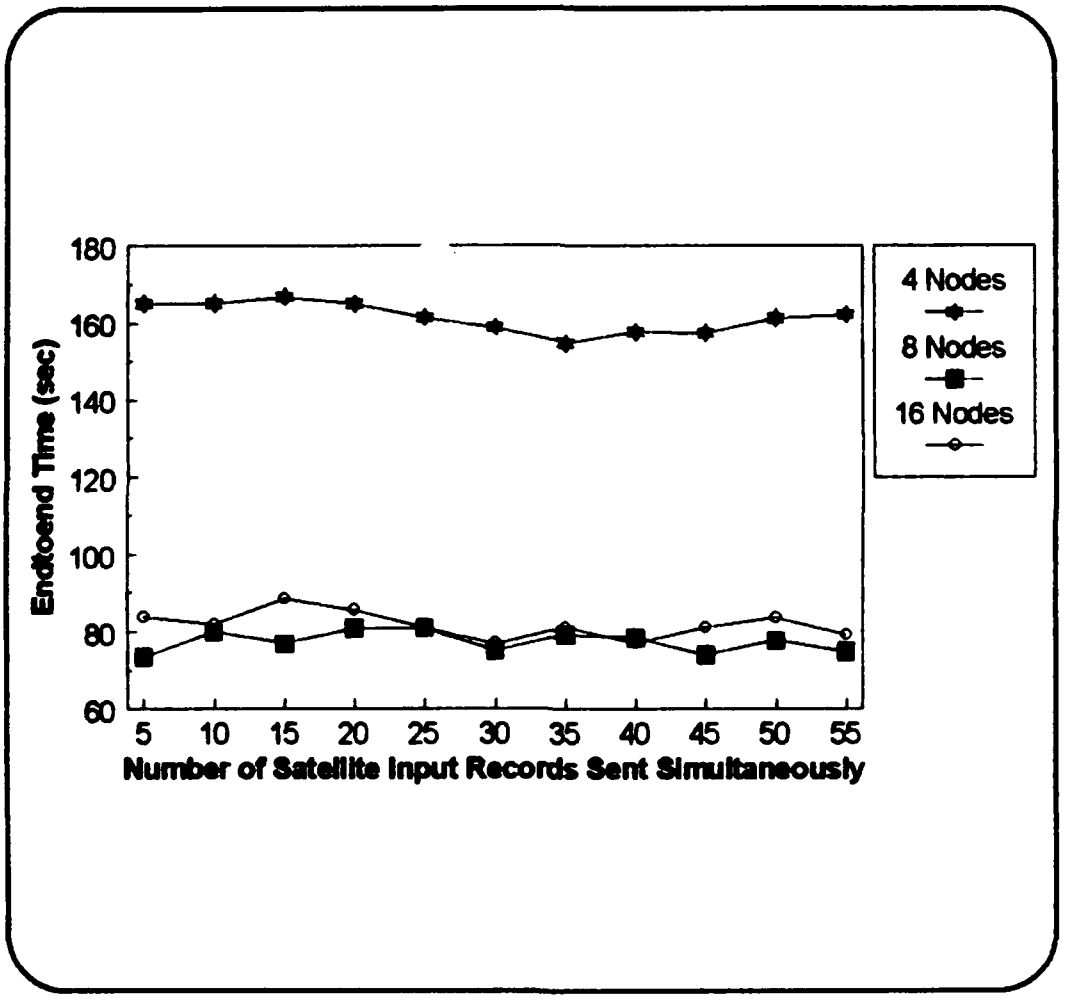


Figure 4.4 Answer Back Model Node Comparison

(2) For the Successive Deal I, a choice of sixteen nodes is superior.

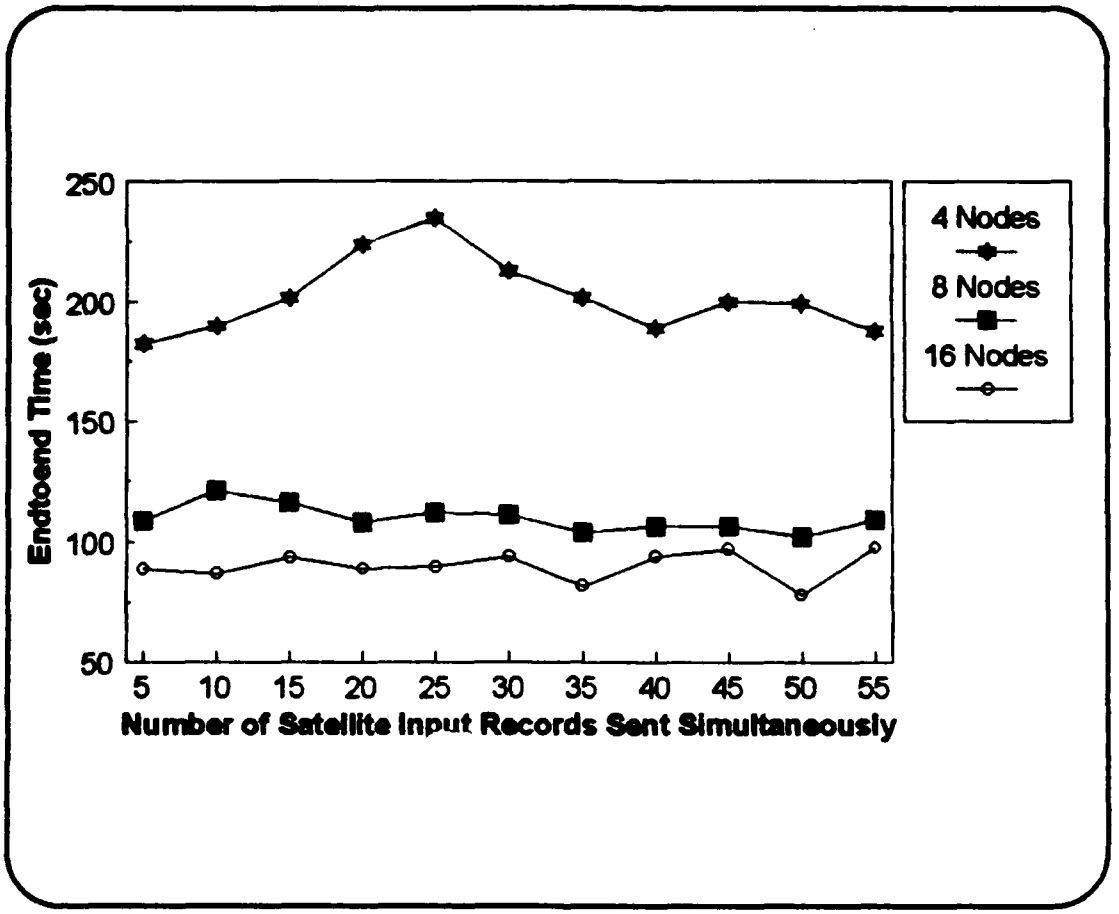


Figure 4.5 Successive Deal Method I - Node Comparison

(3) For the Successive Deal Method II, a choice of sixteen nodes is superior. It is not surprising that sixteen nodes is the best choice for both Successive Deal Methods because both algorithms are very similar; in general, one can note that the number of nodes utilized should decrease the endtoend time. The Successive Deal Method II results can be seen on the next page, Figure 4.6.

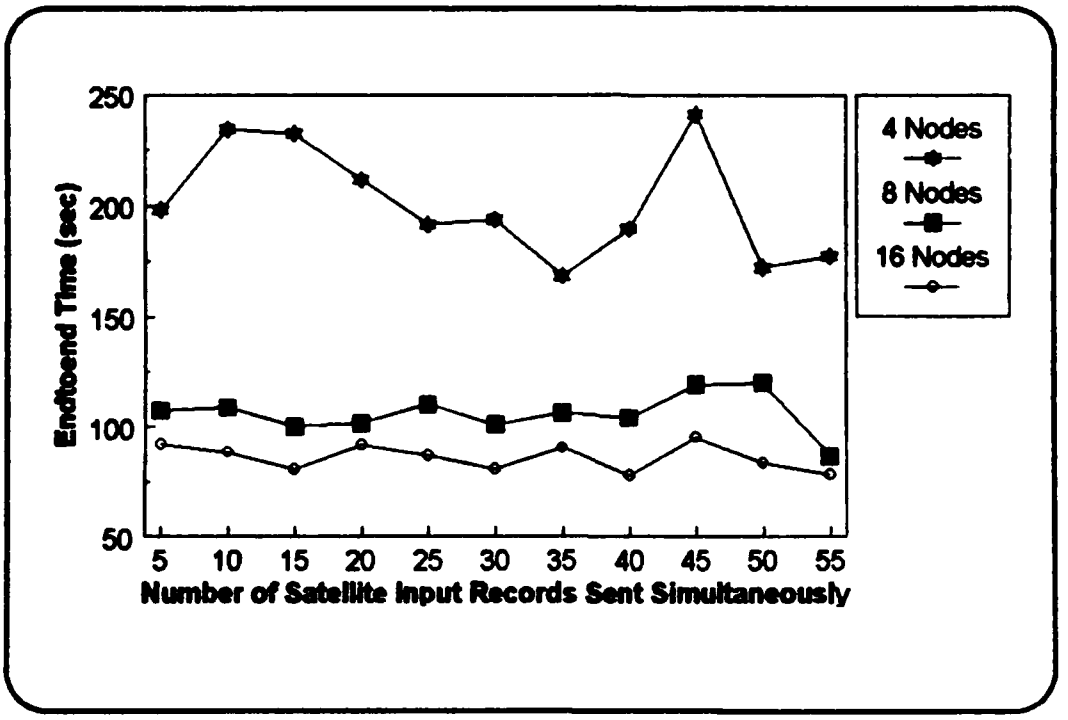


Figure 4.6 Successive Deal Method II - Node Comparison

It is interesting to note that for the Answer Back Model utilizing eight nodes was superior over sixteen nodes for all cases. This could be attributed to the fact that with sixteen nodes the communication time (which was naturally greater in the Answer Back Model) between the master and slaves decreased the advantages of parallelization; whereas, with eight nodes the advantages of parallelization outweighed the disadvantage of the communication time between the master and the slaves.

3. Percent Worker Communication

As one can see from the analysis above, communication time is an important factor in the performance of a given algorithm.

In "PVM Concurrent Computing System: Evolution, Experiences, and Trends" Sunderman, Geist, and Mancheck (p. 7, 1993) state that PVM normally operates in a general purpose networked environment and as a result, raw performance or speedup of a given application is hard to measure. They go on to state that "in such a scenario, most of the focus is on communications overhead."

With communications overhead in mind, the time each worker spent communicating versus the time spent calculating was evaluated. Using average values, the percent of time the worker communicates was calculated as follows:

$$\%\,\text{Worker Communication Time} = \frac{\text{Average Communication Time}}{\text{Average Calculation Time}} \bullet (100\%) \ .$$

The goal was to increase the amount of time a worker spent calculating and decrease the time a worker spent communicating, resulting in a small communication overhead.

a. Model Comparison

For a given number of nodes, the performance of the three models in terms of communication overhead was evaluated and the results are as follows:

(1) Utilizing four nodes for each model produced varied results; in general, the ABM and the SDII were the best choices. The minimum percent worker communication time was attained by the SDII Method when sending 35 satellite input records at a time.

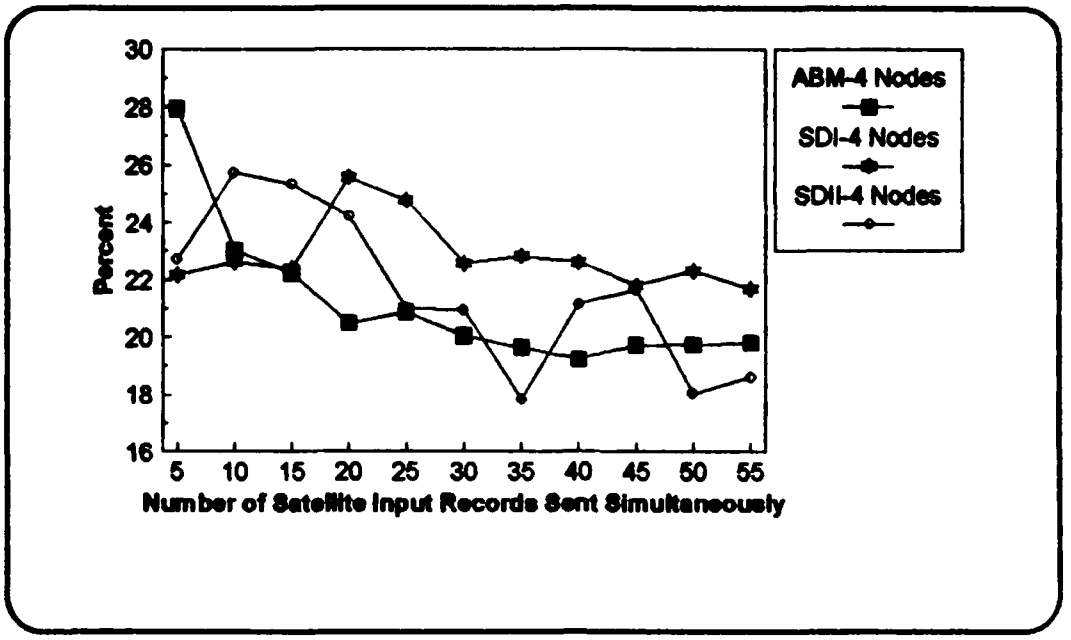


Figure 4.7 Percent Worker Communication For Each Model Using 4 Nodes

(2) When utilizing eight nodes, both Successive Deal Models were, in general, superior over the Answer Back Model. The minimum percent worker communication was attained by SDII when sending 55 satellite input records at a time.

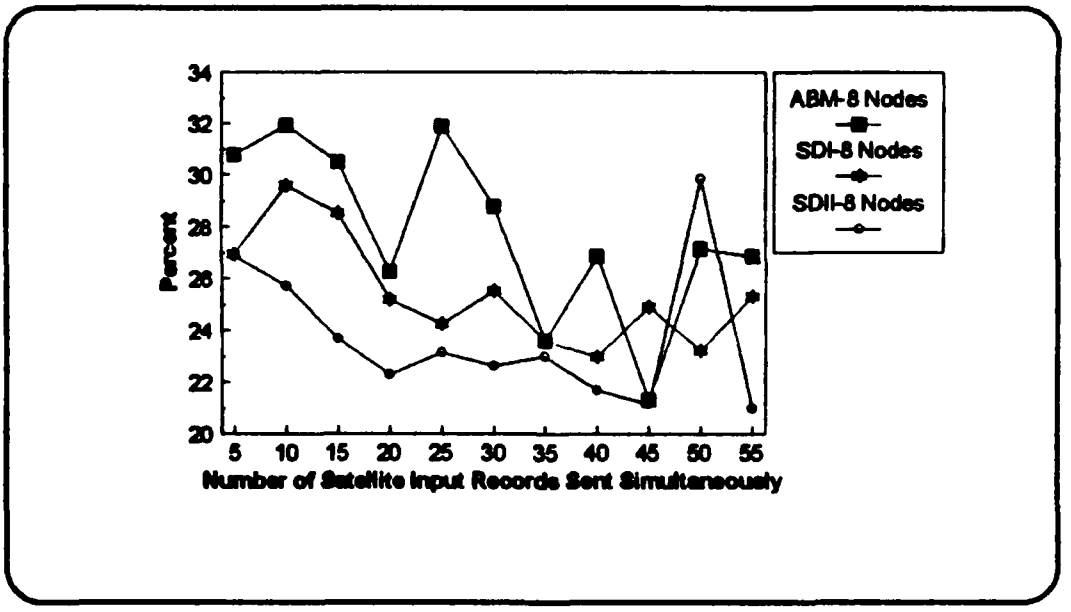


Figure 4.8 Percent Worker Communication For Each Model Using 8 Nodes

(3) When utilizing sixteen nodes, again the Successive Deal Methods were superior over the Answer Back Method. The minimum percent worker communication was attained by the SDII when sending 35 input records at a time.

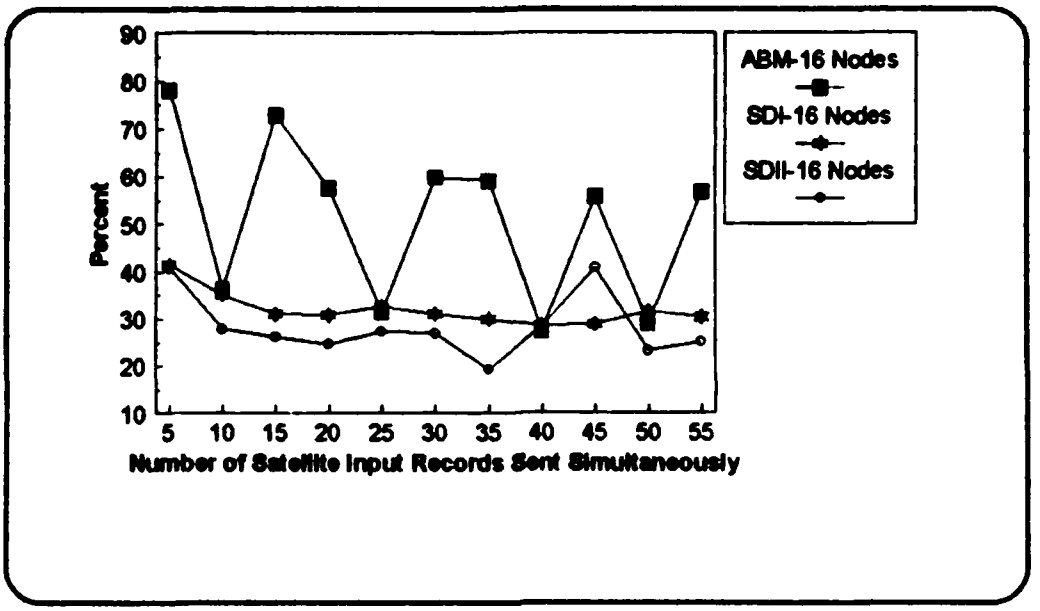


Figure 4.9 Percent Worker Communication For Each Model Using 16 Nodes

The Successive Deal II proved to be the best choice in terms of communication overhead. The Answer Back Method required the additional communication between the master and slaves which increased the communication overhead. The Successive Deal I message size was significantly larger, producing slightly inferior results than the Successive Deal II which continually dealt out small packets of data.

b. Node Comparison

For a given algorithm, the performance of four, eight, and sixteen nodes was evaluated in terms of communication overhead. The results are as follows:

(1) For the ABM, the utilization of 4 nodes was superior.

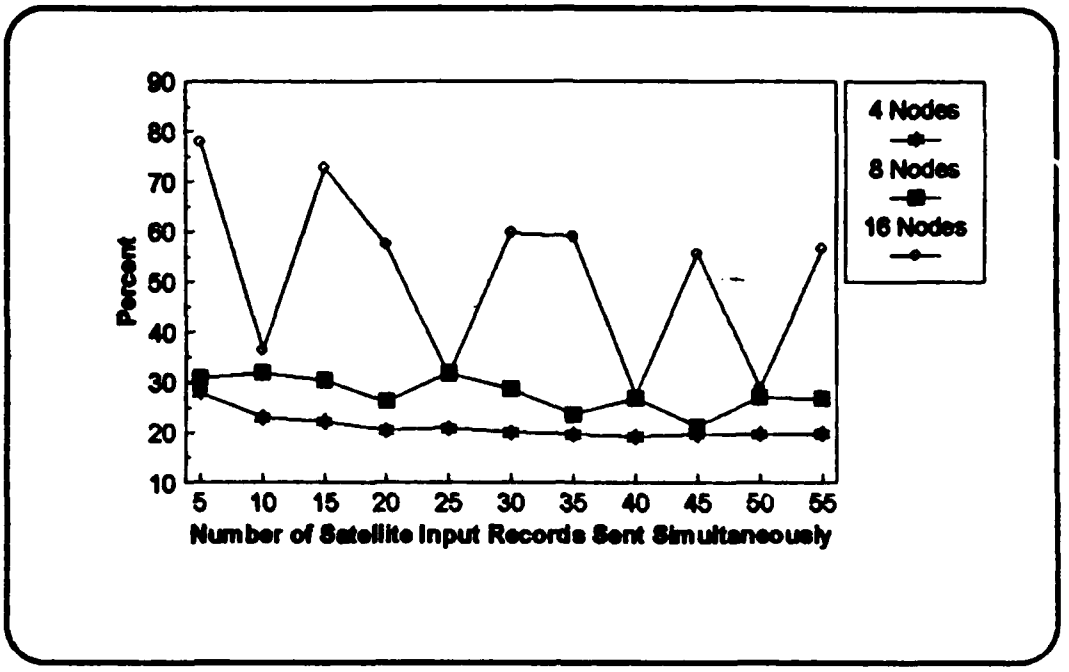


Figure 4.10 ABM Percent Worker Communication - Node Comparison

(2) For the SDI, in general, the utilization of 4 nodes was the best.

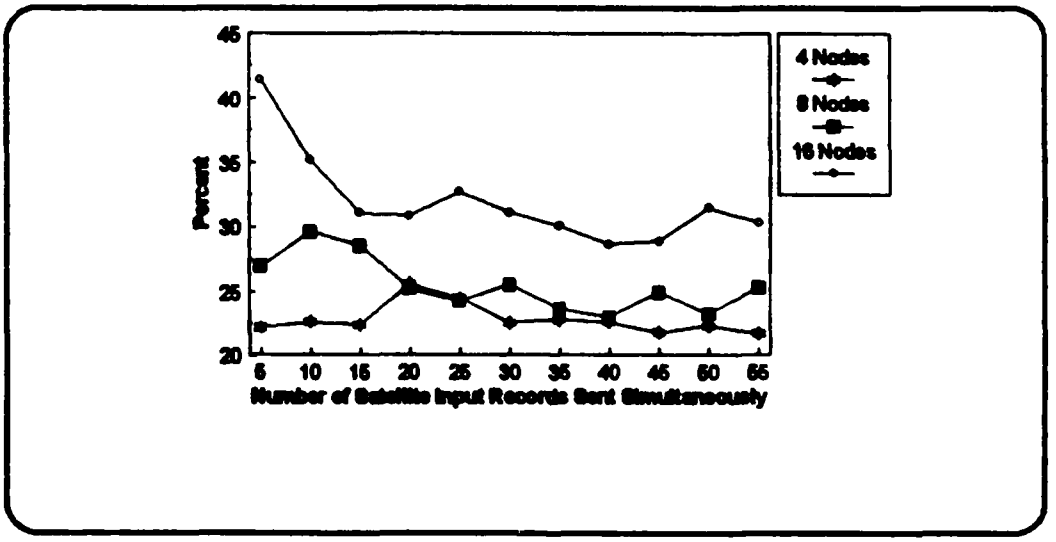


Figure 4.11 SDI Percent Worker Communication - Node Comparison

(3) For SDII the use of four or eight nodes was the best choice.

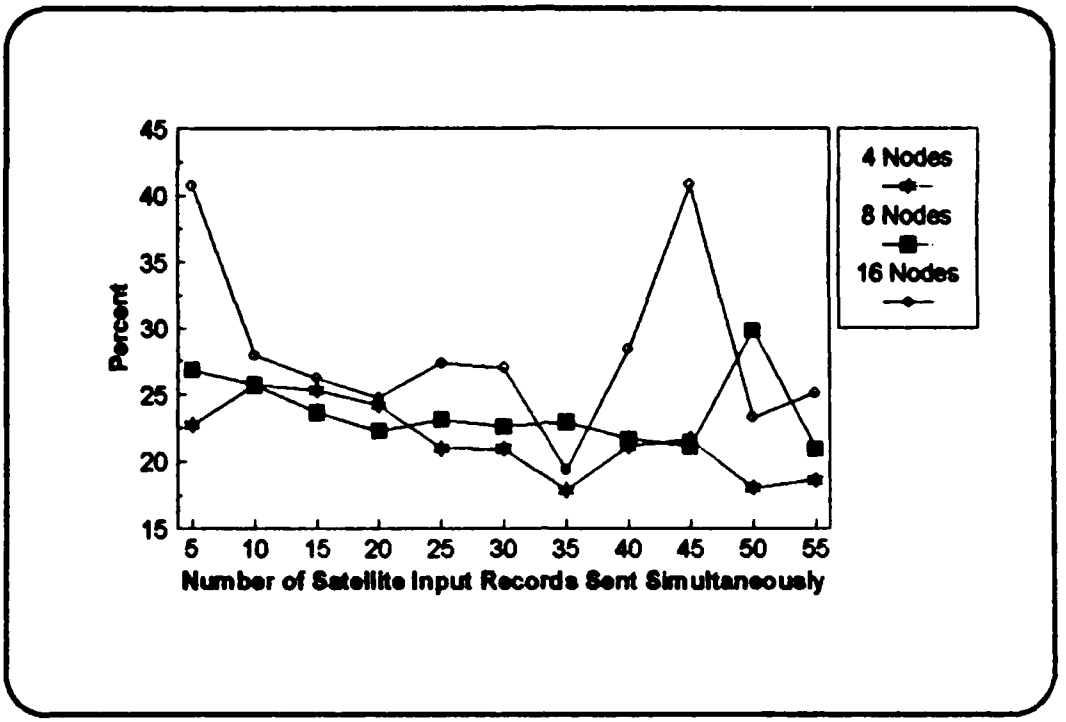


Figure 4.12 SDII Percent Worker Communication - Node Comparison

These results are not surprising due to the fact that for a given algorithm each worker's calculation time is approximately constant (since they all utilize the same input record) and the communication time between the master and slaves is reduced when there are fewer slaves.

4. Speedup

As mentioned earlier, in a general purpose network environment, speedup is hard to measure with a great deal of confidence.

Recall, speedup (S_p) is calculated as follows:

$$S_p = \frac{T_1}{T_p}$$

where T_1 = Endtoend Time on a Single Processor
T_p = Endtoend Time on p Processors

Ideally, the speedup equals "p" the number of processors; however, due to communication costs, sequential bottlenecks, and computational tasks not necessary on a single processor the speedup is less than "p".

With the limitations of speedup results discussed above in mind, the following results were found to be true.

a. Model Comparison

(1) Utilizing four nodes for each model, the ABM was superior.

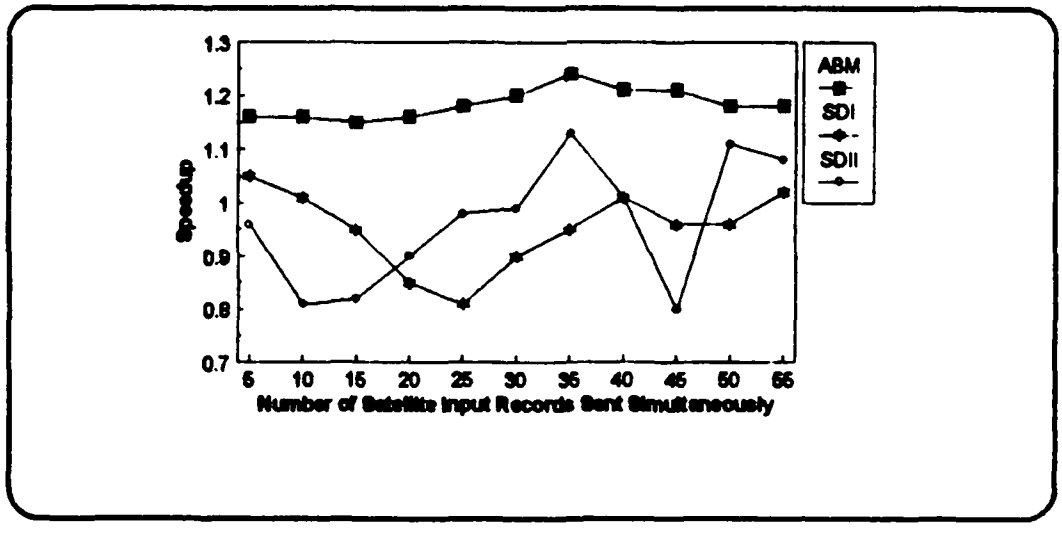


Figure 4.13 Speedup Model Comparison When Using Four Nodes

(2) Utilizing Eight Nodes for each model, the ABM was superior.

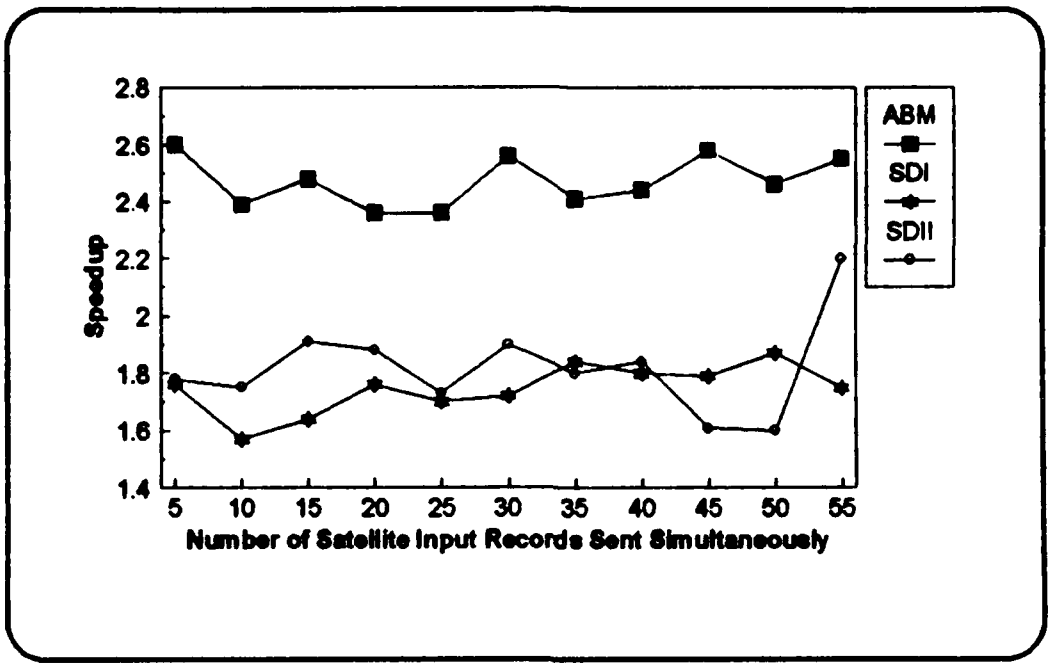


Figure 4.14 Speedup Model Comparison When Using Eight Nodes

(3) Utilizing sixteen nodes, the ABM, in general, was the best.

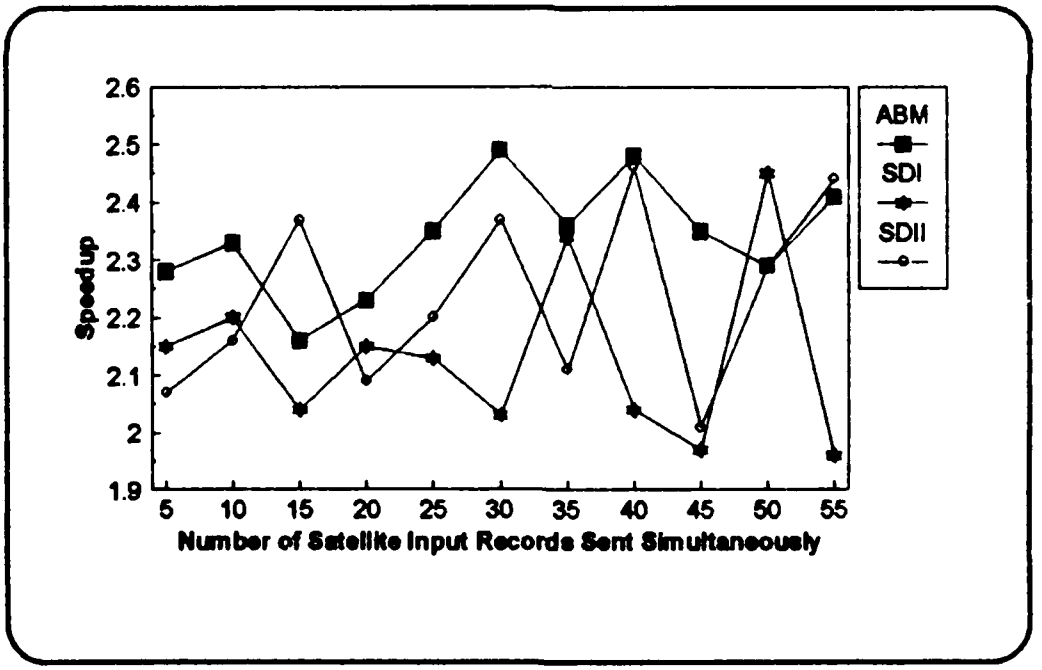


Figure 4.15 Speedup Model Comparison When Using Sixteen Nodes

b. Node Comparison

(1) For the Answer Back Model using 8 or 16 nodes was superior.

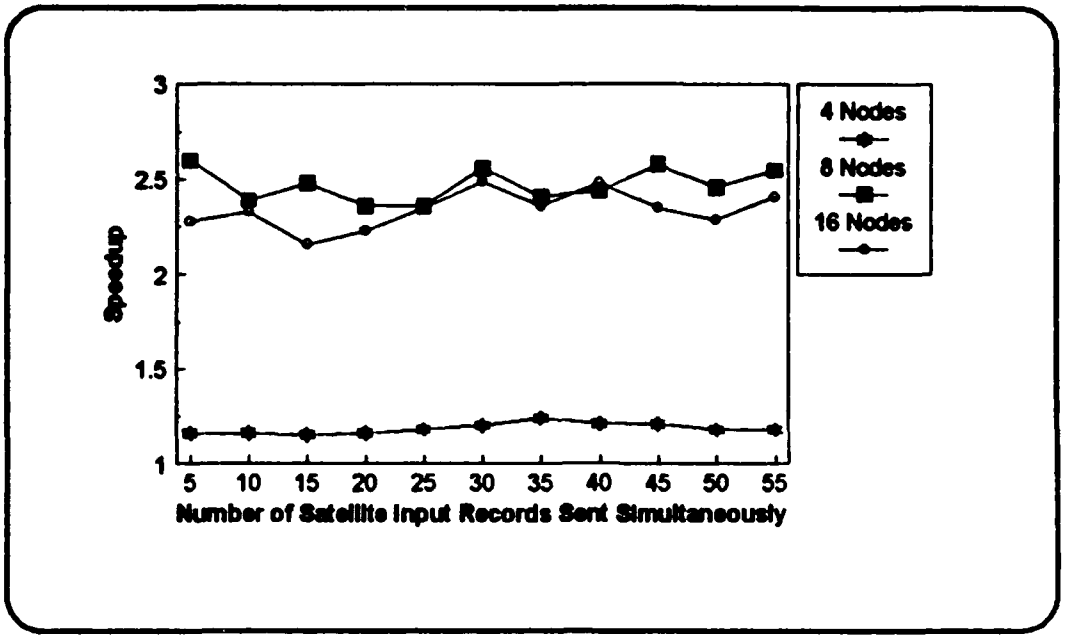


Figure 4.16 Answer Back Model Speedup

(2) For the Successive Deal I the use of 16 nodes was superior.

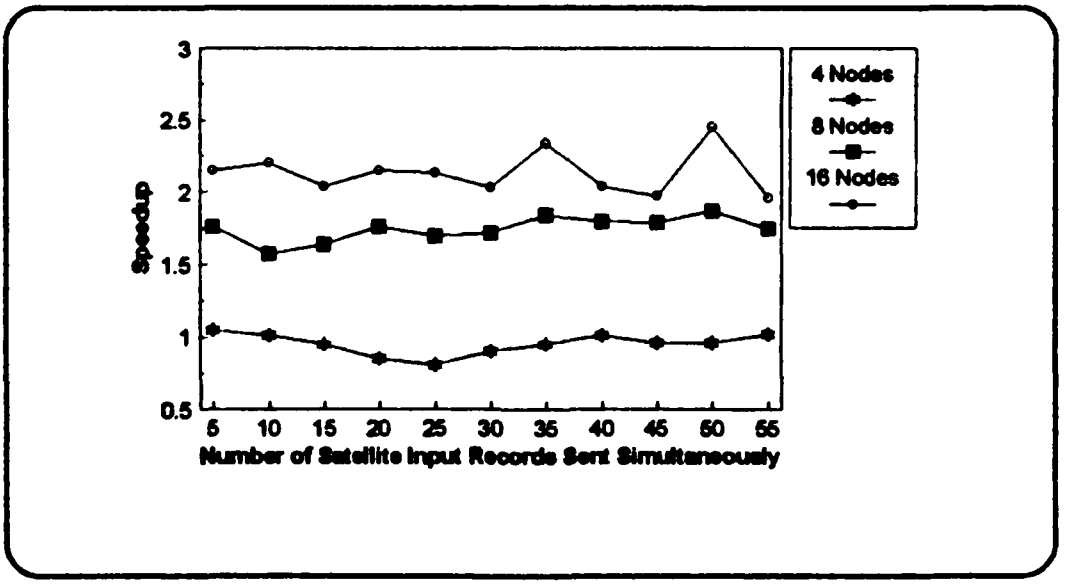


Figure 4.17 Successive Deal I Speedup

(3) For the Successive Deal II, utilizing 16 nodes was superior.

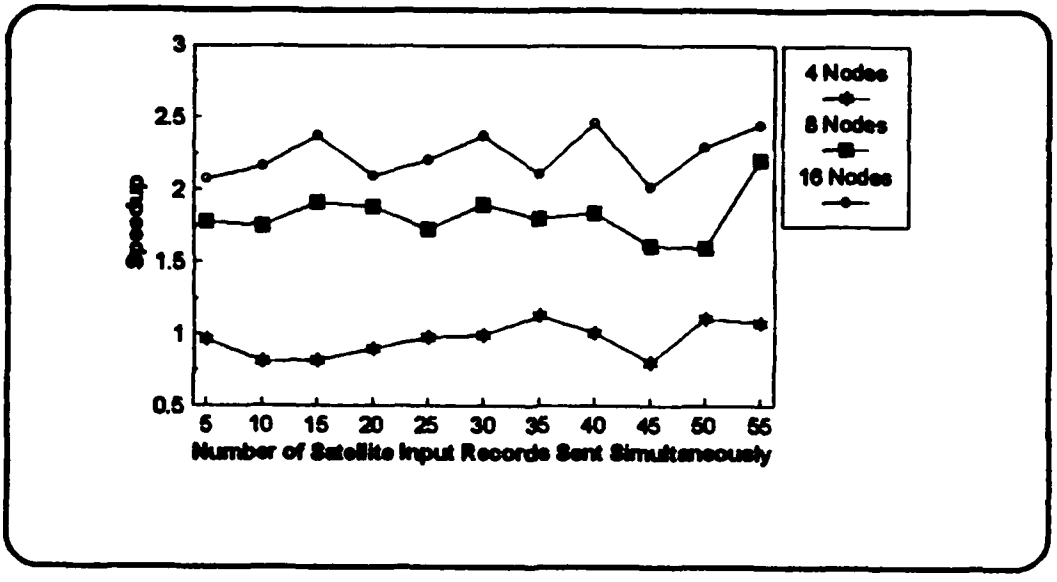


Figure 4.18 Successive Deal II Speedup

These speedup results are directly related to endtoend performance. If one compares figures 4.4-4.6, the endtoend times for each model, and the figures 4.16-4.18 of speedups above an inverse relationship is noted.

5. Efficiency

Recall, Efficiency = $E = \frac{S_p}{p}$

where S_p = Speedup for p processors
p = Number of processors

Thus, the efficiency is a measure of the speedup per processor or how close the actual speedup is to the theoretical speedup (p). The efficiency was evaluated in terms of a comparison of models and a comparison of the node performance for a given model. The results are as follows:

a. Model Comparison

(1) Utilizing 4 nodes the Answer Back Model was superior.

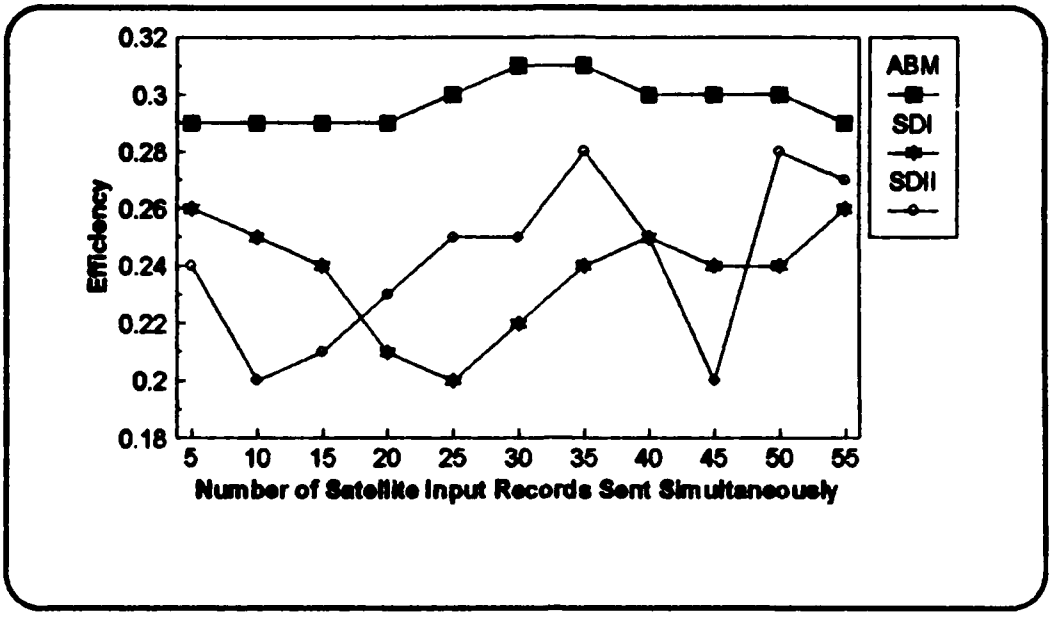


Figure 4.19 Four Node Efficiency Model Comparison

(2) Utilizing 8 nodes, the Answer Back Model was Superior.

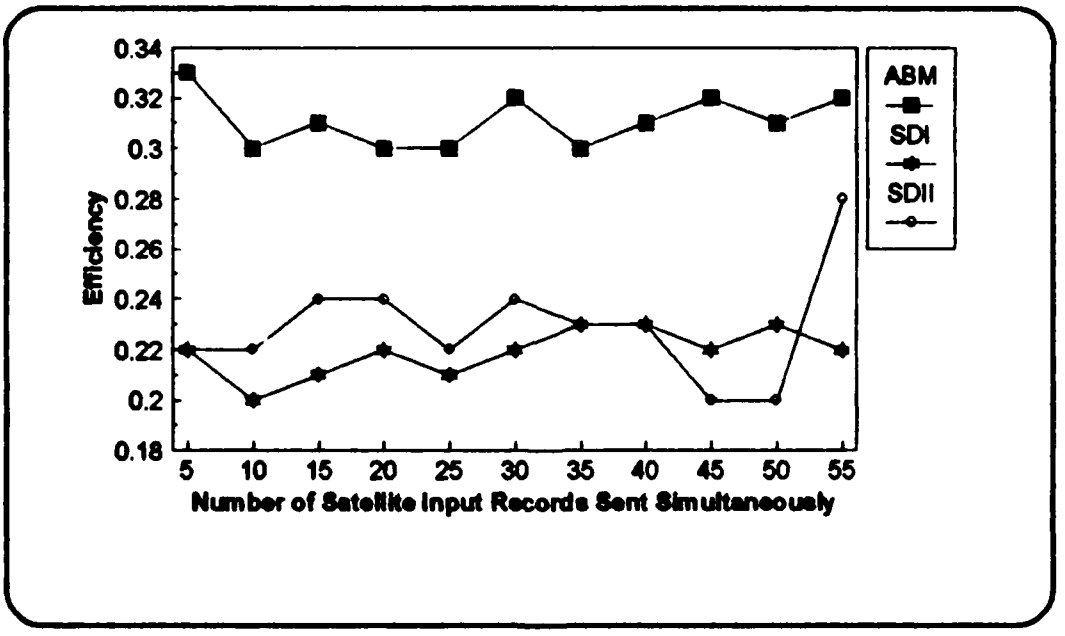


Figure 4.20 Eight Node Efficiency Model Comparison

(3) For sixteen nodes, there was a large fluctuation for all models; however, in general the Answer Back Model was the best choice.

b. Node Comparison

(1) For the ABM, the utilization of 4 or 8 nodes was superior.

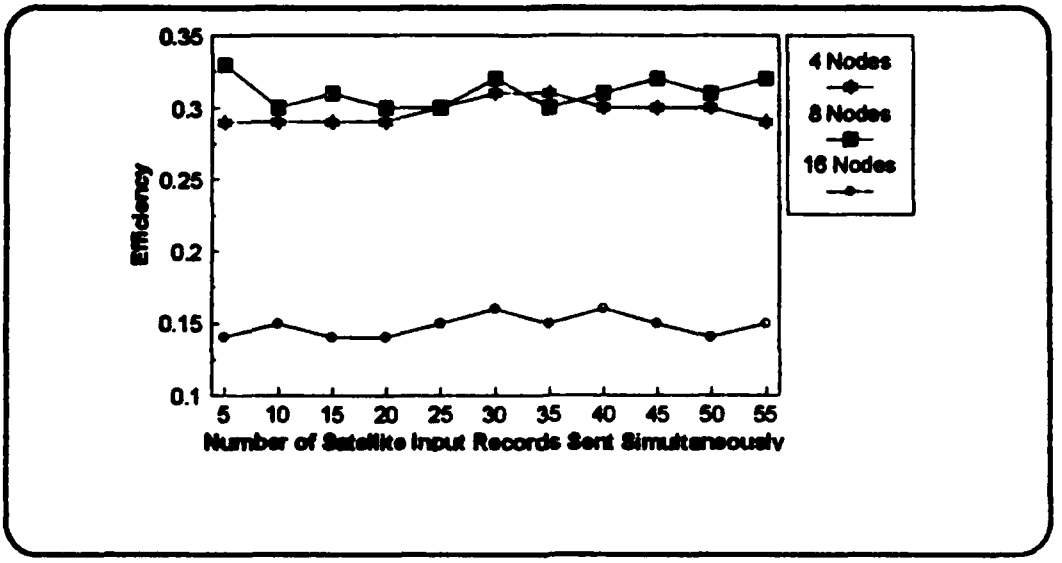


Figure 4.21 Answer Back Model Efficiency

(2) For the SDI using 4 or 8 nodes was the best choice.

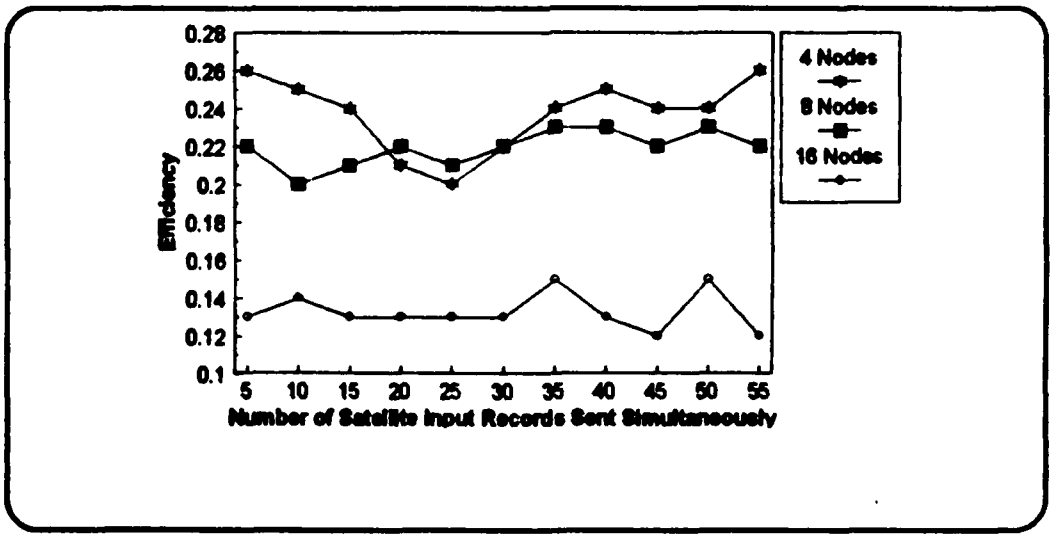


Figure 4.22 Successive Deal I Efficiency

(3) For Successive Deal II , using 4 or 8 nodes was the best choice.

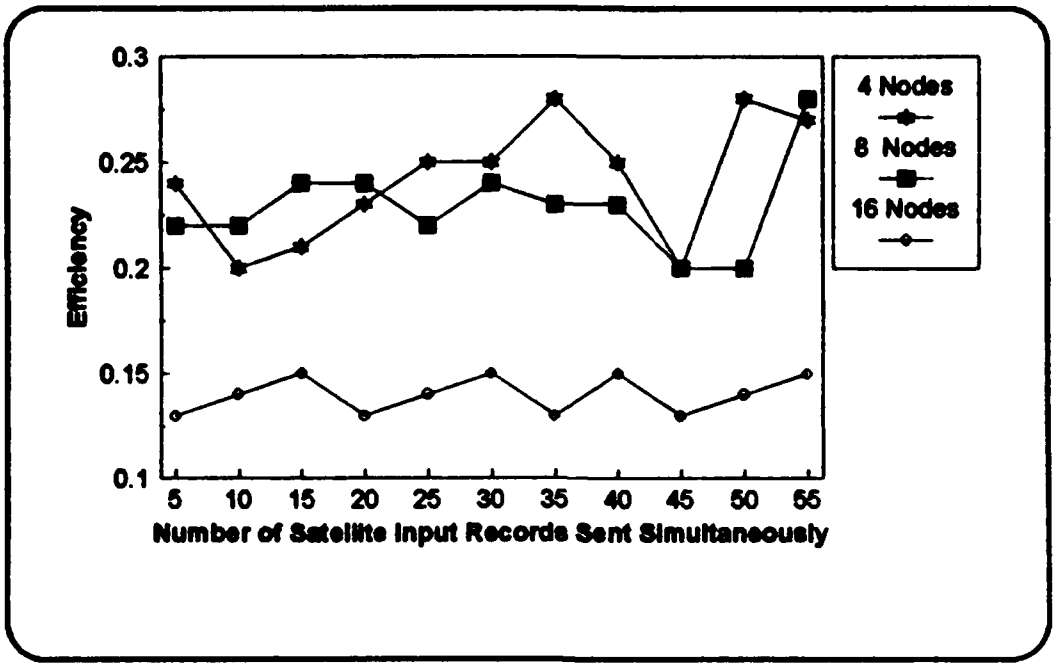


Figure 4.23 Successive Deal II Efficiency

It is important to note that with the use of an open network, there are great fluctuations in the amount of time taken to perform a given task. The execution time depends on the number of current system users and the percentage of the CPU allocated to each user. For example, if one user is running a large application on a given station and another user is using this same station for PVM applications, the execution time will be increased.

In conclusion, considering all factors discussed above, the Answer Back Model was the best algorithm. When using four, eight, or sixteen nodes, the Answer Back Method produced the best Endtoend times, Speedups, and Efficiencies for all size data packets distributed at one time.

The fastest time resulted with the ABM using eight nodes and sending five satellite input records at a time. The utilization of 8 nodes gives the maximum parallelization advantage and the minimum communication overhead. The Answer Back Method required the slaves to notify the master when ready for more data , this reduced the time spent waiting for data; additionally, the fastest workers were the ones that processed more data.

In terms of communication overhead, the Successive Deal II Method was superior to the Successive Deal I and the ABM. The SDII did not have the added communication between the Master and Slaves that was inherent in the Answer Back Method.

No conclusions can be made regarding the best size data packet to send because although sending five input records at a time resulted in the best endtoend time of 73.42 seconds the endtoend time when sending fifty-five records resulted in an endtoend time of 74.85 seconds. Further research would need to be conducted to provide conclusive results on the optimal size data packet to be distributed.

V. CONCLUSIONS

The goal of this thesis is to illustrate how a network of computer workstations is used as a parallel computer to solve a military requirement of tracking 8000 satellites daily.

The Air Force Space Command (AFSPACECOM) satellite computer code ran approximately 2.6 times faster by the parallelization of the code implemented on the Parallel Virtual Machine (PVM) using 8 workstations. PVM is a small software package (~ Mbyte of C source code) that allows a network of computers to appear as a distributed-memory parallel computer.

Many scientists do not use their workstations all the time and when applications are to be run may need more power than a single workstation can provide. The cost of allocating large computing resources to each user is rising daily; thus, the use of PVM or a similar product will be standard in the future.

For military applications, this work illustrates how to use PVM to track satellites using ordinary workstations. A Naval PVM application would be to use a system of workstations located at various enclaves in the ship to track and destroy incoming threats. If the ship took a direct hit in one of its enclaves, the crew would be able to choose unaffected workstations to continue computing power; thus, reducing the vulnerability of the ship.

The AFSPACECOM's Simplified General Perturbation Model Four (SGP4) has been the operational theory since 1976. The SGP4 model uses six classical orbital

elements, a drag factor, and an epoch time to predict a satellite's position and velocity at a future time.

The SGP4 and it's extension, SDP4, are both analytical models. Although the solutions are not as accurate as numerical techniques, they are computationally less expensive. A detailed discussion of the SGP4's mathematical theory can be found in Chapter III.

Currently, D.A. Danielson and B. Neta at the Naval Postgraduate School are documenting and testing a semi-analytical satellite motion model developed by Draper Lab. This will increase the accuracy while decreasing the computational cost. See documentation by Danielson, Early, and Neta (1993) and numerical experiments comparing the semi-analytics to numerical and analytical models by Dyer (1993).

Three algorithms were developed to parallelize the AFSPACECOM code and the performance of each algorithm was tested. All three algorithms use a Master and Slave approach with a separate collector to collect the results and send them back to the Master. The Master distributes the data to the Slaves. The Slaves perform all the calculations necessary to produce the position and velocity vectors for each satellite. The algorithms differed in the manner in which the data is distributed. Each algorithm is tested using four, eight, and sixteen workstations.

The algorithm that required the Slaves to notify the Master when ready for more data resulted in the best times, this method is called the Answer Back Method or ABM. In the ABM, there was less time spent by the Slaves waiting for more data to process

which resulted in the fastest workers processing the most data. When using four, eight, or sixteen workstations, the ABM produced the best total times, speedups, and efficiencies.

One area of further research would include the use of more than sixteen workstations and an algorithm designed to reduce the bottleneck created by the collecting node. Perhaps, the use of two or more collectors would be advantageous. Additionally, further research should be conducted to provide conclusive results on the optimal size data packet to be distributed.

Some of the curves exhibit large fluctuations, this is probably due to changes in the number of users on the system at the time the data was collected. Further research should be conducted to test if the results are reproducible to some extent.

The effect of writing the results to an output file was not considered in this research. Any research conducted in the future should examine the results produced when including the time required to write to an output file.

In conclusion, the result of this thesis confirms that PVM can be used to track orbiting-earth satellites. The use of workstations for parallel processing uses untapped power and decreases the amount of computational time required. As the number of objects to be tracked and the computational power required increases this work will become increasingly more important.

APPENDIX A : SOURCE CODE

```
/*************************************************************************
 *  sat_master_ab.c                              LAST UPDATE:  Oct 5 1993 *
 *  LT S.K. Brewer                                                        *
 * This is the master program for the Answer Back Method. It uses PVM     *
 * to simulate a 2D torus of processors;n+1 slaves are spawned, of        *
 * which n are working nodes and 1 is the collecting node.                *
 * Satellite data is issued to the workers in "Answer Back" fashion,      *
 * sending new data to a working node only when the node is ready.        *
 * Timing data, collecting for statistical purposes only, are placed in   *
 * the file "timing.ans" which will be placed in the directory from       *
 * which this master program is invoked.                                  *
 *************************************************************************/

 #include <stdio.h>              /* INCLUDE STANDARD I/O FUNCTIONS */
 #include "pvm3.h"               /* INCLUDE PVM FUNCTIONS          */
 #include <sys/time.h>
 #include <time.h>
 #include <math.h>
 #include <sys/types.h>

 #define SLAVENAME "at.run"
 int main(argc, argv)            /* GET FILE NAME FROM COMMAND LINE    */
 int       argc;
 char      *argv[];
{
 int       num_nodes=3;       /* NUMBER OF SLAVE NODES   */
 int       num_satdata=15;    /* input data records distributed */
 int       num_elements=22;   /* NUMBER OF elements in each data record */
 double    sat[10000][22];    /* ARRAY OF  satellite input records  */
 int       its,nod,size,delta=5;
 int       num, mytid, i=0, j, k, tids[32], msgtag, reading=1;
 int       numsat=0, collector, leftover, worker, sets, work_nodes,done=0;
 struct    timeval ts[4];          /* Number of time stamps */
 int       who;
 float     endtoend,tcomm,average=0.0,avcoll=0.0,avcomm=0.0,avcalc=0.0;
 float     cmtime, commtime, cctime, calctime, readtime, c_comm,avpcm=0.0;
 float     avpcl=0.0,aa=0.0;
 FILE      *infile, *timing;
 int       msgtag99=99;

 gettimeofday(&ts[0],(struct timeval *)0);/* BEGIN READING DATA FILE */

 /* OPEN DATA FILE  */
 if ((infile = fopen(argv[1], "r")) == NULL)
 {   printf("infile = %s did not open\n", argv[1]);
  exit(1);
 }
```

```
/*    READ ENTIRE DATA FILE AT ONCE    */
while(reading != EOF)
{   if ((reading = fscanf(infile, "%lf", &sat[numsat][0])) != EOF)
     for (j=1; j<num_elements; ++j)
        fscanf(infile, "%lf", &sat[numsat][j]);
   numsat=numsat+1;      /* COUNT NUMBER OF SATELLITES IN DATA FILE */
}
fclose(infile);
numsat=numsat-1;
gettimeofday(&ts[1],(struct timeval *)0);    /* END READING DATA FILE */
/* SET UP FILE FOR TIMING STATISTICS    */
timing = fopen("timing.ans", "a");
readtime = (ts[1].tv_sec-ts[0].tv_sec)*1000000+ts[1].tv_usec-ts[0].tv_usec;
fprintf(timing, "Time to read data file = %ld microseconds\n", readtime);
for(size=0; size<55; size +=delta)
{
 num_satdata = size + 5;
 for(nod=0; nod<3; ++nod)
 {
 if(nod == 0)
  num_nodes = 3;
 else
   if(nod == 1)
      num_nodes = 7;
   else num_nodes = 15;

 fprintf(timing, "sats,nodes,    endtoend    collector_comm
 worker_comm worker_calc\n");
 fprintf(timing,"%d     %d\n",num_satdata,num_nodes);


  for(its=0; its<1; ++its)
  {
  gettimeofday(&ts[2], (struct timeval *)0);/* BEGIN END TO END TIME*/
  /********** ENROLL IN PVM  **********/

    mytid = pvm_mytid();

  /* START UP SLAVE TASKS     */
     num=pvm_spawn(SLAVENAME, (char**)0, 0, "", num_nodes, tids);
     collector=tids[0];

    /*    SEND SLAVES THIER INDICES INTO THE TID ARRAY   */
    msgtag=1;
    for (i=0; i<num_nodes; ++i)
    {   pvm_initsend(PvmDataRaw);
        pvm_pkint(&i,1,1);
       if (i==0)
        pvm_pkint(&numsat, 1, 1);  /* TELL COLLECTOR NUMBER OF SATS */
       else
        pvm_pkint(&collector, 1, 1);/* TELL WORKERS COLLECTOR'S ADDRESS*/
        pvm_send( tids[i], msgtag);
    }
```

```
/* SEND SETS OF SATELLITE DATA TO WORKERS, WAITING FOR ANSWER BACK */
msgtag=2;
k=0;
work_nodes=num_nodes-1;
sets=numsat/num_satdata;
leftover=numsat-sets*num_satdata;
i=0;
for(j=1;j<num_nodes; ++j)      /* DEAL ONE SET OF SATELLITES TO EACH WORKER */
{   pvm_initsend(PvmDataRaw);
    pvm_pkint(&num_satdata,1,1);
    for (k=0; k<num_satdata; ++k)
    {  pvm_pkdouble(sat[i], num_elements,1);
       i=i+1;
    }
    pvm_send(tids[j], msgtag);
    sets=sets-1;
}
while(sets>0)  /* DEAL REMAINING SETS TO WORKERS AS THE NODES BECOME FREE */
{   pvm_initsend(PvmDataRaw);
    pvm_pkint(&num_satdata,1,1);
    for (k=0; k<num_satdata; ++k)
    {  pvm_pkdouble(sat[i], num_elements,1);
       i=i+1;
    }
    sets=sets-1;
    pvm_recv(-1, msgtag99);
    pvm_upkint(&who,1,1);
    pvm_send(tids[who],msgtag);
}
if (leftover>0)   /* SEND LEFTOVERS TO WHOEVER IS READY NEXT */
{   pvm_initsend(PvmDataRaw);
    pvm_pkint(&leftover,1,1);
    for (k=0; k<leftover; ++k)
    {   pvm_pkdouble(sat[i], num_elements,1);
        i=i+1;
    }
    pvm_recv(-1,msgtag99);
    pvm_upkint(&who,1,1);
    pvm_send(tids[who],msgtag);
}

pvm_initsend(PvmDataRaw);
pvm_pkint(&done, 1, 1); /*TELL WORKERS NO MORE DATA IS COMING*/
pvm_mcast(tids, num_nodes, msgtag);

msgtag=5;/* RECEIVE PROGRAM COMPLETE SIGNAL FROM COLLECTOR */
pvm_recv(-1,msgtag);
/* COMPLETE END TO END TIME      */
gettimeofday(&ts[3],(struct timeval *)0);
```

```
/* GATHER TIMING STATISTICS FROM SLAVES */
msgtag=4;
for (i=0; i<num_nodes; ++i)
{   pvm_recv(-1,msgtag);
    pvm_upkint(&who,1,1);
    if (who == 0)  /* TIMES FROM COLLECTOR */
    {
         pvm_upklong(&c_comm,1,1);
    }
    else
                   /* TIMES FROM WORKERS  */
    {
        pvm_upklong(&cctime,1,1);
        calctime=calctime+cctime;
        pvm_upklong(&cmtime,1,1);
        commtime=commtime+cmtime;

    }
}
pvm_exit();
/* COMPUTE OVERALL TIMING STATISTICS */
endtoend=(float)(ts[3].tv_sec-ts[2].tv_sec)*1000000+
         (float)ts[3].tv_usec-(float)ts[2].tv_usec;
    /*convert to seconds*/
    c_comm=c_comm/1.0E6;
    endtoend=endtoend/1.0E6;
    commtime=commtime/1.0E6;
    calctime=calctime/1.0E6;
    /* TOTAL TIME*/
   average = average + endtoend;
   avcoll = avcoll + c_comm;       /*collector communication time*/
   avcomm = avcomm + commtime;     /*worker communication time*/
   avcalc = avcalc + calctime;     /*worker calculation time*/
   fprintf(timing, "                %6.2f                %6.2f
         %6.2f         %6.2f\n",endtoend,c_comm,commtime,calctime);
}

 average = average/its;
 avcoll = avcoll/its;
 avcomm = avcomm/its;
 avcalc = avcalc/its;
 avpcm=avcomm/(num_nodes-1);
 avpcl=avcalc/(num_nodes-1);
 aa=(avpcm/avpcl)*100;
 /* print results to output file - not shown in this code */
}

}
  fclose(timing);
  printf("ENTIRE SEQUENCE COMPLETE");

}
```

```
/*************************************************************
 *                                                           *
 *  sat_slave_ab.c    LAST UPDATE: 05 OCT 1993               *
 *                                 Susan Brewer              *
 * This is the slave program for the Answer Back Method.     *
 * It uses PVM to simulate a 2D torus of processors.         *
 *  The slave with index 0 will be the collecting node.      *
 * This program "answers back" for more data.                *
 * The Fortran sub-routine "sgp4m"is called to perform the   *
 *  calculations for orbit prediction                        *
 *************************************************************/

#include "pvm3.h"                /* INCLUDE PVM FUNCTIONS      */
#include <stdio.h>
#include <sys/time.h>
#include <time.h>
#include <math.h>
#include <sys/types.h>

main()
{

 double   results[7*100+1]; /* ARRAY OF RESULTS  */
 int      num_elements=22;  /* FIELDS IN INPUT SATELLITE RECORD */
 double   sat_data[22];     /* ONE SATELLITE INPUT RECORD       */
 int      max=8000, sats=1;
 int      sat_no;
 int      i,j, k, t, r_length;  /* COUNTERS                     */
 int      tids[32];               /* ARRAY OF PROCESSOR IDS     */
 int      mytid, numnode;         /* MY PROCESSOR ID            */
 int      me, collector;   /* MY INDEX INTO THE TIDS ARRAY      */
 int      master,msgtag, msgtag2=2, msgtag3=3, msgtag99=99;
 struct   timeval ts[4];
 int      res_sets=0;
 float    s=0.0, u=0.0, totaltime, calc, comm;

extern   sgp4m_ (); /* EXTERNAL SUB-ROUTINE FOR ORBIT PREDICTION */

 mytid = pvm_mytid();               /* ENROLL IN PVM            */
 master=pvm_parent();

 /* RECEIVE MY INDEX AND COLLECTOR'S TID FROM MASTER            */
 gettimeofday(&ts[0],(struct timeval *)0);
 msgtag = 1;
 pvm_recv( -1, msgtag );
 pvm_upkint(&me, 1, 1);    /* GET MY INDEX IN THE ARRAY OF TIDs */
 pvm_upkint(&collector, 1, 1):    /* GET THE COLLECTING NODE'S TID*/
```

```
 if(me == 0)                    /* IF I AM THE COLLECTING NODE:       */
 {
  for(i=0; i< max; ++i)
  {
   pvm_recv( -1, msgtag3);
   pvm_upkint(&sat_no, 1, 1);/* RECEIVE RESULT SETS  */
   pvm_upkint(&r_length, 1, 1);
   pvm_upkdouble(results, r_length, 1);
}
msgtag=5;           /* TELL MASTER ALL RESULTS HAVE BEEN received */
pvm_initsend(PvmDataRaw);
pvm_send(master, msgtag);
}
else            /* If I AM A WORKING NODE:                          */
{
 while(sats>0)  /* REPEAT UNTIL MASTER SENDS DONE SIGNAL            */
 {  pvm_recv(-1, msgtag2);
  pvm_upkint(&sats, 1, 1);
  for (i=0; i<sats; ++i)
  {   pvm_upkdouble(sat_data, num_elements ,1);
      sat_no=(int)sat_data[1];
      gettimeofday(&ts[2],(struct timeval *)0);
      sgp4m_ (sat_data, results); /* CALL SUB-ROUTINE*/
      gettimeofday(&ts[3],(struct timeval *)0);
      s=s+ts[3].tv_sec-ts[2].tv_sec;
      u=u+ts[3].tv_usec-ts[2].tv_usec;
      r_length=7*(int)results[0]+1; /* NUMBER OF RESULTS RECORDS */
      pvm_initsend(PvmDataRaw);
      pvm_pkint( &sat_no, 1, 1 ); /* SATELLITE NUMBER */
      pvm_pkint( &r_length, 1, 1);
      pvm_pkdouble( results, r_length, 1 ); /*PACK */
      pvm_send(collector, msgtag3);          /* SEND */
 pvm_initsend(PvmDataRaw); /*TELL MASTER I'M READY FOR MORE DATA */
 pvm_pkint(&me,1,1);
 pvm_send(master, msgtag99);
  }
}/* TIMING STATISTICS TO BE SENT TO MASTER  */
gettimeofday(&ts[1],(struct timeval *)0);
totaltime=(float)(ts[1].tv_sec-ts[0].tv_sec)*1000000+
          (float)ts[1].tv_usec-(float)ts[0].tv_usec;
   calc = s*1000000 + u;
   comm = totaltime - calc;
    msgtag=4;
    pvm_initsend(PvmDataRaw);
    pvm_pkint(&me, 1,1);
  if(me == 0)
   {
   pvm_pklong(&totaltime,1,1);
   }
 else
   {
  pvm_pklong(&calc,1,1);pvm_pklong(&comm,1,1);
    }
  pvm_send(master,msgtag);
    pvm_exit();
}
```

```
/*****************************************************************
 *                                                               *
 *  sat_master_SDI.c          LAST UPDATE:  Oct 12 1993          *
 *                                          LT S.K. BREWER       *
 * This is the master program for the Successive Deal Method I.  *
 * It uses PVM to simulate a 2D torus of processors;n+1 slaves   *
 * are spawned, of which n are working nodes and 1 is the        *
 * collecting node. Satellite data is issued to the workers by   *
 * first dealing one data package (num_satdata) to each worker,  *
*then deal 1/(2*working nodes)times the number of data sets     *
 * left(num_sets) .Followed by a final deal of equal packets to  *
*each worker. Any leftover records are sent last. Timing data,  *
*collecting for statistical purposes only, are placed in the    *
 * file "timing" which will be placed in the directory from which *
 * this master program is invoked.                               *
 *****************************************************************/
#include <stdio.h>          /* INCLUDE STANDARD I/O FUNCTIONS     */
#include "pvm3.h"                    /* INCLUDE PVM FUNCTIONS     */
#include <sys/time.h>
#include <time.h>
#include <math.h>
#include <sys/types.h>

#define SLAVENAME "t.run"
 int main(argc, argv)       /* GET FILE NAME FROM COMMAND LINE   */
 int       argc;
 char      *argv[];
 {
   int       num_nodes;          /* NUMBER OF SLAVE NODES */
   int       num_satdata;        /* NUMBER OF input data records*/
   int       num_elements=22;    /* NUMBER OF elements    */
   double    sat[10000][22];         /* ARRAY  */
   int       its,nod,size,delta=5;
   int       num, mytid, i=0, j=0, k=0, s=0, tids[32], msgtag;
   int       numsat=0, collector, reading=1;
   int       leftover=0,setsleft=0,worker=0, sets=0,num_sets=0;
   int       work_nodes=0,done=0;
   struct    timeval ts[4];          /* Time Stamps required */
   int       who;
   float    endtoend=0.0,tcomm=0.0,average=0.0,avcoll=0.0;
   float    avcomm=0.0,avcalc=0.0,c_comm,avpcm=0.0,avpcl=0.0,aa=0.0;
   float    cmtime, commtime, cctime, calctime, readtime;
   FILE     *infile, *timing;

    /* BEGIN READING DATA FILE */
   gettimeofday(&ts[0],(struct timeval *)0);
   /* OPEN DATA FILE  */
   if ((infile = fopen(argv[1], "r")) == NULL)
    {   printf("infile = %s did not open\n", argv[1]);
        exit(1);
    }
```

```
     /*    READ ENTIRE DATA FILE AT ONCE    */
           while(reading != EOF)
     (if ((reading = fscanf(infile, "%lf", &sat[numsat][0])) != EOF)
         for (j=1; j<num_elements; ++j)
            fscanf(infile, "%lf", &sat[numsat][j]);
            numsat=numsat+1; /* NUMBER OF SATELLITES IN DATA FILE */
     }
      fclose(infile);
      numsat=numsat-1;
     /* END READING DATA FILE */
    gettimeofday(&ts[1],(struct timeval *)0);
     /* SET UP FILE FOR TIMING STATISTICS    */
     timing = fopen("timing", "a");
     readtime = (ts[1].tv_sec-ts[0].tv_sec)*1000000+
                 ts[1].tv_usec-ts[0].tv_usec;
 fprintf(timing,"Time to read data file = %ld microseconds\n",readtime);
 for(size=0; size<55; size +=delta)
 {
   num_satdata = size + 5;
   for(nod=0; nod<3; ++nod)
   {
      if(nod == 0)
       num_nodes = 3;
      else
      if(nod == 1)
       num_nodes = 7;
      else num_nodes = 15;
  for(its=0; its<10; ++its)
  {
  leftover=0;
  setsleft=0;
  sets=0;
  num_sets=0;
 gettimeofday(&ts[2], (struct timeval *)0);/* BEGIN END TO END TIME*/

/********** ENROLL IN PVM  ***********/

 mytid = pvm_mytid();

/* START UP SLAVE TASKS     */
  num=pvm_spawn(SLAVENAME, (char**)0, 0, "", num_nodes, tids);
  collector=tids[0];
 /*    SEND SLAVES THIER INDICES INTO THE TID ARRAY  */
 msgtag=1;
 for (i=0; i<num_nodes; ++i)
 {   pvm_initsend(PvmDataRaw);
     pvm_pkint(&i,1,1);
     if (i==0)
        pvm_pkint(&numsat, 1, 1);
     else
        pvm_pkint(&collector, 1, 1);
     pvm_send( tids[i], msgtag);
 }
```

```
/* SEND SETS OF SATELLITE DATA TO WORKERS */
msgtag=2;
k=0;
work_nodes=num_nodes-1;
sets=numsat/num_satdata;
leftover=numsat-sets*num_satdata;
i=0;
for(j=1;j<num_nodes; ++j) /* DEAL SET OF SATS TO EACH WORKER */
{   pvm_initsend(PvmDataRaw);
    pvm_pkint(&num_satdata,1,1);
    for (k=0; k<num_satdata; ++k)
    {  pvm_pkdouble(sat[i], num_elements,1);
       i=i+1;
    }
    pvm_send(tids[j], msgtag);
}
    sets=sets-work_nodes;
    num_sets=sets/(2*work_nodes);

for(j=1; j<num_nodes; ++j)    /* Deal 1/2p records */
{
    for(s=0; s<num_sets; ++s)
      {
        pvm_initsend(PvmDataRaw);
        pvm_pkint(&num_satdata,1,1);
        for (k=0; k<num_satdata; ++k)
        {
          pvm_pkdouble(sat[i], num_elements,1);
          i=i+1;
        }
        pvm_send(tids[j],msgtag);

      }
}
  sets=sets-(num_sets*work_nodes);
  num_sets=sets/work_nodes;
  setsleft=sets-(num_sets*work_nodes);
  /* Deal remaining records in equal packets */
  for(j=1; j<num_nodes; ++j)
  {
    for(s=0; s<num_sets; ++s)
      {
        pvm_initsend(PvmDataRaw);
        pvm_pkint(&num_satdata,1,1);
        for (k=0; k<num_satdata; ++k)
        {
          pvm_pkdouble(sat[i], num_elements,1);
          i=i+1;
        }
        pvm_send(tids[j],msgtag);

      }
}
```

```
if(setsleft>0)      /*send leftover sets*/
{
    for(s=0; s<setsleft; ++s)
      {
        pvm_initsend(PvmDataRaw);
        pvm_pkint(&num_satdata,1,1);
        for (k=0; k<num_satdata; ++k)
        {
          pvm_pkdouble(sat[i], num_elements,1);
            i+1;
        }
        pvm_send(tids[1],msgtag);
      }
}
if(leftover>0)     /* send leftover records*/
{
    pvm_initsend(PvmDataRaw);
    pvm_pkint(&leftover,1,1);
    for (j=0; j<leftover; ++j)
    {
        pvm_pkdouble(sat[i], num_elements,1);
        i=i+1;
    }
    pvm_send(tids[1],msgtag);
}
pvm_initsend(PvmDataRaw);
pvm_pkint(&done, 1, 1);  /* TELL WORKERS NO MORE DATA IS COMING*/
pvm_mcast(tids,num_nodes, msgtag);

msgtag=5;/* RECEIVE PROGRAM COMPLETE SIGNAL FROM COLLECTOR   */
pvm_recv(-1,msgtag);

gettimeofday(&ts[3],(struct timeval *)0); /* END TO END TIME*/

/* GATHER TIMING STATISTICS FROM SLAVES */
msgtag=4;
for (i=0; i<num_nodes; ++i)
{   pvm_recv(-1,msgtag);
    pvm_upkint(&who,1,1);
    if (who == 0)  /* TIMES FROM COLLECTOR */
    {
     pvm_upklong(&c_comm,1,1);/* TIME COLLECTOR COMM */
    }
    else                                          /* TIMES FROM WORKERS  */
    {
        pvm_upklong(&cctime,1,1);
        calctime=calctime+cctime;
        pvm_upklong(&cmtime,1,1);
        commtime=commtime+cmtime;
    }
}
pvm_exit();
```

```
  /* COMPUTE OVERALL TIMING STATISTICS */

      /*COMM TIME*/

    endtoend=(float)(ts[3].tv_sec-ts[2].tv_sec)*1000000+
             (float)ts[3].tv_usec-(float)ts[2].tv_usec;

      /*convert to seconds*/
      c_comm=c_comm/1.0E6;
      endtoend=endtoend/1.0E6;
      commtime=commtime/1.0E6;
      calctime=calctime/1.0E6;
      /* TOTAL TIME*/
     average = average + endtoend;
     avcoll = avcoll + c_comm;   /*collector communication time*/
     avcomm = avcomm + commtime; /*worker communication time*/
     avcalc = avcalc + calctime; /*worker calculation time*/
  endtoend = 0.0;calctime = 0.0;commtime = 0.0;c_comm   = 0.0;
 }
  average = average/its;
  avcoll = avcoll/its;
  avcomm = avcomm/its;
  avcalc = avcalc/its;
  avpcm=avcomm/(num_nodes-1);
  avpcl=avcalc/(num_nodes-1);
  aa=(avpcm/avpcl)*100;
  /* Print results to output file - not shown in this code */
  average=0.0;
  avcoll=0.0;
  avcomm=0.0;
  avcalc=0.0;
  avpcm=0.0;
  avpcl=0.0;
  aa=0.0;
 }

}
  fclose(timing);
  printf("ENTIRE SEQUENCE COMPLETE - results have been appended to timing");

}
```

```
/*************************************************************
 *                                                           *
 *  sat_slave_SDI.c          LAST UPDATE: 12 OCT 1993        *
 *                                        LT S.K. BREWER     *
 *  This is the slave program for Successive Deal I.         *
 *   It uses PVM to simulate a 2D torus of processors.       *
 *   The slave with index 0 will be the collecting node.     *
 * The Fortran sub-routine "sgp4m" is called to perform      *
 *  the calculations for orbit prediction.                   *
 *************************************************************/
#include "pvm3.h"              /* INCLUDE PVM FUNCTIONS      */
#include <stdio.h>
#include <sys/time.h>
#include <time.h>
#include <math.h>
#include <sys/types.h>

main()
{

  double   results[7*100+1];    /* ARRAY OF RESULTS */
  int      num_elements=22;     /* NUMBER OF FIELDS */
  double   sat_data[22];        /* ONE SATELLITE INPUT RECORD*/
  int      sats=1,maxsats;
  int      sat_no;
  int      i,j, k, t, r_length; /* COUNTERS                   */
  int      tids[32];             /* ARRAY OF PROCESSOR IDS    */
  int      mytid, numnode;      /* MY PROCESSOR ID            */
  int      me, collector;   /* MY INDEX INTO THE TIDS ARRAY */
  int      master,msgtag, msgtag2=2, msgtag3=3;
  struct   timeval ts[4];
  float    s=0.0, u=0.0, totaltime, calc, comm;

  extern   sgp4m_ (); /*    TERNAL SUB-ROUTINE */

   mytid = pvm_mytid();    /* ENROLL IN PVM     */
  master=pvm_parent();

  /* RECEIVE MY INDEX AND COLLECTOR'S TID FROM MASTER        */
  gettimeofday(&ts[0],(struct timeval *)0);
  msgtag = 1;
  pvm_recv( -1, msgtag );
  pvm_upkint(&me, 1, 1);  /*GET MY INDEX IN THE ARRAY OF TIDs*/
  pvm_upkint(&collector, 1, 1);
```

```
if(me == 0)                /* IF I AM THE COLLECTING NODE */
{
 maxsats=collector;
   for(i=0; i<maxsats; ++i)
   {
    pvm_recv( -1, msgtag3);
    pvm_upkint(&sat_no, 1, 1);/* RECEIVE RESULT Sets */
    pvm_upkint(&r_length, 1, 1);
   pvm_upkdouble(results, r_length, 1);
   }
    msgtag=5; /* TELL MASTER ALL RESULTS HAVE BEEN received */
    pvm_initsend(PvmDataRaw);
    pvm_send(master, msgtag);
   }
  else        /* If I AM A WORKING NODE */
   {
    while(sats>0) /* REPEAT UNTIL MASTER SENDS DONE SIGNAL */
      {pvm_recv(-1, msgtag2);
       pvm_upkint(&sats, 1, 1);
       for (i=0; i<sats; ++i)
       {   pvm_upkdouble(sat_data, num_elements ,1);
       sat_no=(int)sat_data[1];
       gettimeofday(&ts[2],(struct timeval *)0);
       sgp4m_ (sat_data, results);       /* CALL SUB-ROUTINE */
       gettimeofday(&ts[3],(struct timeval *)0);
       s=s+ts[3].tv_sec-ts[2].tv_sec;
       u=u+ts[3].tv_usec-ts[2].tv_usec;
       r_length=7*(int)results[0]+1;
       pvm_initsend(PvmDataRaw);
       pvm_pkint( &sat_no, 1, 1 );       /* SATELLITE NUMBER*/
       pvm_pkint( &r_length, 1, 1);
       pvm_pkdouble( results, r_length, 1 );    /*PACK */
       pvm_send(collector, msgtag3);           /* SEND */
    }
 }
)/* TIMING STATISTICS TO BE SENT TO MASTER  */
gettimeofday(&ts[1],(struct timeval *)0);
totaltime=(float)(ts[1].tv_sec-ts[0].tv_sec)*1000000+
         (float)ts[1].tv_usec-(float)ts[0].tv_usec;
calc = s*1000000 + u;
comm = totaltime - calc;
msgtag=4;
pvm_initsend(PvmDataRaw);
pvm_pkint(&me, 1,1);
if(me == 0)
{
  pvm_pklong(&totaltime,1,1);
}
else
{
   pvm_pklong(&calc,1,1); pvm_pklong(&comm,1,1);
}
   pvm_send(master,msgtag);pvm_exit();
}
```

```
/*************************************************************
 *                                                           *
 *  sat_master_SDII.c    LAST UPDATE:  Oct 13 1993           *
 *                                        LT S.K. BREWER     *
 * This is the master program for the Successive             *
 * Deal II. It uses PVM to simulate a 2D torus of            *
 * processors; n+1 slaves are spawned, of which n            *
 * are working nodes and 1 is the collecting node.           *
 * Satellite data is issued to the workers by                *
 * constantly dealing out equal size data packs.             *
 * Timing data, collecting for statistical purposes          *
 * are placed in the file "timrr" which will be              *
 * placed in the directory from which this master            *
 * program is invoked.                                       *
 *************************************************************/

#include <stdio.h>  /* INCLUDE STANDARD I/O FUNCTIONS */
#include "pvm3.h"             /* INCLUDE PVM FUNCTIONS */
#include <sys/time.h>
#include <time.h>
#include <math.h>
#include <sys/types.h>

#define SLAVENAME "t.run"
int main(argc, argv) /* GET FILE NAME FROM COMMAND LINE */
int       argc;
char      *argv[];
{
  int       num_nodes;         /* NUMBER OF SLAVE NODES */
  int       num_satdata;       /* # input records dealt */
  int       num_elements=22;
  double    sat[10000][22];          /* ARRAY  */
  int       its,nod,size,delta=5;
  int       num, mytid, i=0, j, k, tids[32], msgtag;
  int       numsat=0, collector, leftover, worker, sets;
  int       work_nodes, done=0,reading=1;
  struct    timeval ts[4];    /* Number of time stamps * /
  int       who;
  float     endtoend,tcomm,average=0.0,avcoll=0.0;
  float     avcomm=0.0,a  alc=0.0, readtime, c_comm,avpcm=0.0;
  float     cmtime,commtime,cctime,calctime,avpcl=0.0,aa=0.0;
  FILE      *infile, *timing;
  /* BEGIN READING DATA FILE */
  gettimeofday(&ts[0],(struct timeval *)0);
 /* OPEN DATA FILE  */
 if ((infile = fopen(argv[1], "r")) == NULL)
 {   printf("infile = %s did not open\n", argv[1]);
     exit(1);
 }
```

```
/*    READ ENTIRE DATA FILE AT ONCE   */
  while(reading !  EOF)
   {   if ((reading = fscanf(infile, "%lf", &sat[numsat][0])) != EOF)
       for (j=1; j<num_elements; ++j)
       fscanf(infile, "%lf", &sat[numsat][j]);
       numsat=numsat+1;/* COUNT NUMBER OF SATELLITES IN DATA FILE */
   }
         fclose(infile);
         numsat=numsat-1;
         /* END READING DATA FILE */
       gettimeofday(&ts[1],(struct timeval *)0);
     /* SET UP FILE FOR TIMING STATISTICS   */
       timing = fopen("timrr", "a");
 readtime = (ts[1].tv_sec-ts[0].tv_sec)*1000000+
            ts[1].tv_usec-ts[0].tv_usec;

 for(size=0; size<55; size +=delta)
 {
    num_satdata = size + 5;

    for(nod=0; nod<3; ++nod)
    {
      if(nod == 0)
      num_nodes = 3;
   else
       if(nod == 1)
          num_nodes = 7;
       else num_nodes = 15;

for(its=0; its<10; ++its)
{ /* BEGIN END TO END TIME */
gettimeofday(&ts[2], (struct timeval *)0);
/********** ENROLL IN PVM  **********/

mytid = pvm_mytid();

/* START UP SLAVE TASKS    */
 num=pvm_spawn(SLAVENAME, (char**)0, 0, "", num_nodes, tids);
 collector=tids[0];

/*    SEND SLAVES THIER INDICES INTO THE TID ARRAY  */
msgtag=1;
for (i=0; i<num_nodes; ++i)
{   pvm_initsend(PvmDataRaw);
    pvm_pkint(&i,1,1);
    if (i==0)
       pvm_pkint(&numsat, 1, 1);
    else
       pvm_pkint(&collector, 1, 1);
    pvm_send( tids[i], msgtag);
}
```

```
/* SEND SETS OF SATELLITE DATA TO WORKERS */
msgtag=2;
k=0;
work_nodes=num_nodes-1;
sets=numsat/num_satdata;
leftover=numsat-sets*num_satdata;

for (i=0; i<sets; ++i)
{   worker = i-(i/work_nodes)*work_nodes+1;

    pvm_initsend(PvmDataRaw);
    pvm_pkint(&num_satdata, 1, 1);
    for(j=0; j<num_satdata; ++j)
    {  pvm_pkdouble(sat[k], num_elements, 1);
       k=k+1;
    }
    pvm_send( tids[worker], msgtag);
}
if (leftover>0) /*  SEND LEFTOVERS */
{   pvm_initsend(PvmDataRaw);
    pvm_pkint(&leftover, 1, 1);
    for(j=0; j<leftover; ++j)
    {   pvm_pkdouble(sat[k], num_elements, 1);
        k=k+1;
    }
    pvm_send(tids[work_nodes], msgtag);
}
pvm_initsend(PvmDataRaw);
/* TELL WORKERS NO MORE DATA IS COMING */
pvm_pkint(&done, 1, 1);
for(j=1; j< num_nodes; ++j)
{
pvm_send(tids[j], msgtag);
}
msgtag=5;/* RECEIVE PROGRAM COMPLETE SIGNAL FROM COLLECTOR*/
pvm_recv(-1,msgtag);
/* COMPLETE END TO END TIME      */
gettimeofday(&ts[3],(struct timeval *)0);
/* GATHER TIMING STATISTICS FROM SLAVES */
msgtag=4;
for (i=0; i<num_nodes; ++i)
{   pvm_recv(-1,msgtag);
    pvm_upkint(&who,1,1);
```

```
            if (who == 0)  /* TIMES FROM COLLECTOR */
            {
             pvm_upklong(&c_comm,1,1);     /* TIME COLLECTOR SPENT COMMUNICATING */
            }
            else
            /* TIMES FROM WORKERS  */
            { pvm_upklong(&cctime,1,1); /* TIME SPENT CALCULATING RESULTS */
              calctime=calctime+cctime;
              pvm_upklong(&cmtime,1,1);  /* TIME SPENT COMMUNICATING OR WAITING */
              commtime=commtime+cmtime;
            }
          }
          pvm_exit();

          /* COMPUTE OVERALL TIMING STATISTICS */
             /*COMM TIME*/
           endtoend=(float)(ts[3].tv_sec-ts[2].tv_sec)*1000000+
                    (float)ts[3].tv_usec-(float)ts[2].tv_usec;
            /*convert to seconds*/
            c_comm=c_comm/1.0E6;
            endtoend=endtoend/1.0E6;
            commtime=commtime/1.0E6;
            calctime=calctime/1.0E6;
            /* TOTAL TIME*/

            average = average + endtoend;
            avcoll = avcoll + c_comm;        /*collector communication time*/
            avcomm = avcomm + commtime;      /*worker communication time*/
            avcalc = avcalc + calctime;      /*worker calculation time*/
          endtoend = 0.0;calctime = 0.0;commtime = 0.0;c_comm   = 0.0;
      }
          average = average/its;
          avcoll = avcoll/its;
          avcomm = avcomm/its;
          avcalc = avcalc/its;
          avpcm=avcomm/(num_nodes-1);
          avpcl=avcalc/(num_nodes-1);
          aa=(avpcm/avpcl)*100;
          /* Print statistics to output file - not shown in code */
          average=0.0;
          avcoll=0.0;
          avcomm=0.0;
          avcalc=0.0;
          avpcm=0.0;
          avpcl=0.0;
          aa=0.0;
    }

  }
          fclose(timing);
          printf("ENTIRE SEQUENCE COMPLETE ");

}
```

```
/*************************************************************
*                                                            *
*  sat_slave_SDII.c          LAST UPDATE: 13 OCT 1993        *
*                                          LT S.K. BREWER    *
*  This is the slave program for Successive Deal I.          *
*   It uses PVM to simulate a 2D torus of processors.        *
*   The slave with index 0 will be the collecting node.      *
* The Fortran sub-routine "sgp4m" is called to perform       *
*  the calculations for orbit prediction.                    *
*************************************************************/
#include "pvm3.h"              /* INCLUDE PVM FUNCTIONS      */
#include <stdio.h>
#include <sys/time.h>
#include <time.h>
#include <math.h>
#include <sys/types.h>

main()
{

  double   results[7*100+1];   /* ARRAY OF RESULTS */
  int      num_elements=22;    /* NUMBER OF FIELDS */
  double   sat_data[22];       /* ONE SATELLITE INPUT RECORD*/
  int      sats-i,maxsats;
  int      sat_no;
  int      i,j, k, t, r_length; /* COUNTERS                  */
  int      tids[32];            /* ARRAY OF PROCESSOR IDS    */
  int      mytid, numnode;      /* MY PROCESSOR ID           */
  int      me, collector;   /* MY INDEX INTO THE TIDS ARRAY */
  int      master,msgtag, msgtag2=2, msgtag3=3;
  struct   timeval ts[4];
  float    s=0.0, u=0.0, totaltime, calc, comm;

  extern   sgp4m_ (); /* EXTERNAL SUB-ROUTINE */

   mytid = pvm_mytid();   /* ENROLL IN PVM    */
  master=pvm_parent();

  /* RECEIVE MY INDEX AND COLLECTOR'S TID FROM MASTER       */
  gettimeofday(&ts[0],(struct timeval *)0);
  msgtag = 1;
  pvm_recv( -1, msgtag );
  pvm_upkint(&me, 1, 1);  /*GET MY INDEX IN THE ARRAY OF TIDs*/
  pvm_upkint(&collector, 1, 1);
```

```
if(me == 0)              /* IF I AM THE COLLECTING NODE */
{
 maxsats=collector;
   for(i=0; i<maxsats; ++i)
   {
    pvm_recv( -1, msgtag3);
    pvm_upkint(&sat_no, 1, 1);/* RECEIVE RESULT Sets */
    pvm_upkint(&r_length, 1, 1);
   pvm_upkdouble(results, r_length, 1);
   }
    msgtag=5; /* TELL MASTER ALL RESULTS HAVE BEEN received */
    pvm_initsend(PvmDataRaw);
    pvm_send(master, msgtag);
   }
  else       /* If I AM A WORKING NODE */
   {
    while(sats>0) /* REPEAT UNTIL MASTER SENDS DONE SIGNAL */
      {pvm_recv(-1, msgtag2);
       pvm_upkint(&sats, 1, 1);
       for (i=0; i<sats; ++i)
       {   pvm_upkdouble(sat_data, num_elements ,1);
       sat_no=(int)sat_data[1];
       gettimeofday(&ts[2],(struct timeval *)0);
       sgp4m_ (sat_data, results);      /* CALL SUB-ROUTINE */
       gettimeofday(&ts[3],(struct timeval *)0);
       s=s+ts[3].tv_sec-ts[2].tv_sec;
       u=u+ts[3].tv_usec-ts[2].tv_usec;
       r_length=7*(int)results[0]+1;
       pvm_initsend(PvmDataRaw);
       pvm_pkint( &sat_no, 1, 1 );      /* SATELLITE NUMBER*/
       pvm_pkint( &r_length, 1, 1);
       pvm_pkdouble( results, r_length, 1 );   /*PACK */
       pvm_send(collector, msgtag3);          /* SEND */
    }
 }
)/* TIMING STATISTICS TO BE SENT TO MASTER  */
gettimeofday(&ts[1],(struct timeval *)0);
totaltime=(float)(ts[1].tv_sec-ts[0].tv_sec)*1000000+
         (float)ts[1].tv_usec-(float)ts[0].tv_usec;
calc = s*1000000 + u;
comm = totaltime - calc;
msgtag=4;
pvm_initsend(PvmDataRaw);
pvm_pkint(&me, 1,1);
if(me == 0)
{
  pvm_pklong(&totaltime,1,1);
}
else
{
   pvm_pklong(&calc,1,1); pvm_pklong(&comm,1,1);
}
   pvm_send(master,msgtag);pvm_exit();
}
```

```
/*****************************************************************
 *                                                               *
 *   seq.c    LT S.K. BREWER      OCT 25 93                      *
 *                                                               *
 * This is a sequential version of the satellite orbit           *
 *  prediction program using SGP4.                               *
 *                                                               *
 *****************************************************************/

#include <stdio.h>          /* INCLUDE STANDARD I/O FUNCTIONS*/
#include <sys/time.h>
#include <time.h>
#include <math.h>
#include <sys/types.h>

int main(argc, argv)       /* GET FILE NAME FROM COMMAND LINE*/
int       argc;
char      *argv[];
{
    int       iterations=50;
    int       num_elements=22;
    double    sat[32000][22];/*ARRAY OF SATELLITE INPUT DATA */
    int       its;    /* NUMBER OF ITERATIONS OF THE PROGRAM */
    int       i=0, j, k, t, reading=1;
    int       numsat=0;
    struct    timeval ts[4]; /* Number of Time Stamps Required*/
    float     endtoend=0.0,average=0.0
    long      readtime;
    int       sat_no;
    double    results[7*100+1];
    FILE      *infile, *timing;

    extern sgp4m_ ();
   /* BEGIN READING DATA FILE */
  gettimeofday(&ts[0],(struct timeval *)0); /* OPEN DATA FILE*/
  if ((infile = fopen(argv[1], "r")) == NULL)
   {   printf("infile = %s did not open\n", argv[1]);
       exit(1);
   }


  /*    READ ENTIRE DATA FILE AT ONCE   */
   while(reading != EOF)
   { if ((reading = fscanf(infile,"%lf",&sat[numsat][0])) != EOF)
 for (j=1; j<num_elements; ++j)
     fscanf(infile, "%lf", &sat[numsat][j]);
 numsat=numsat+1;   /* COUNT NUMBER OF SATELLITES IN DATA FILE */
 }
   fclose(infile);
   numsat=numsat-1;
 gettimeofday(&ts[1],(struct timeval *)0); /* END READING DATA FILE */
```

```
    /* SET UP FILE FOR TIMING STATISTICS   */
    timing = fopen("timing.seq", "a");
   readtime = (ts[1].tv_sec-ts[0].tv_sec)*1000000+
   ts[1].tv_usec-ts[0].tv_usec;
 for(its=0; its<iterations; ++its)
 {
  gettimeofday(&ts[2],(struct timeval *)0);
  for (i=0; i<numsat; ++i)
 {   sat_no=(int)sat[i][1];
   sgp4m_ (sat[i], results);
}
gettimeofday(&ts[3],(struct timeval *)0);
endtoend=(float)(ts[3].tv_sec-ts[2].tv_sec)*1000000+
         (float)ts[3].tv_usec-(float)ts[2].tv_usec;
/* convert to seconds */
   endtoend=endtoend/1.0E6;
/* write results to timing output file */

fprintf(timing, "\n Endtoend time (sec) =  %6.2f\n",endtoend);

/* Total Time */
average=average+endtoend;
}
average=average/its;
fprintf(timing, "\n Average Endtoend time (sec)= %6.2f\n", average);
fclose(timing);

printf("\nENTIRE SEQUENCE COMPLETE ");
}
```

List of References

1. Brower, D., "Solution of the Problem of Artificial Satellite Theory Without Drag", *Astronomical Journal*, v.64, pp. 378-397, 1959.
2. Danielson, D.A., Early, L.W., and Neta B., " Semianalytic Satellite Theory (SST): Mathematical Algorithms, Naval Postgraduate School, Monterey, California, Technical Report , 1993.
3. Dongarra, Geist, Mancheck, Sunderman, "Integrated PVM Framework Supports Heterogeneous Network Computing", *Computers in Physics*, pp. 166-175 March/April 1993.
4. Dyar, Walt, "Comparison of Orbit Propagators in the Research and Development Goddard Trajectory Determination System", *Masters Thesis*, Naval Postgraduate School, September 1993.
5. Eichelberger and Provencher, "A Survivable AEGIS Combat System Model", *EC4820 Project*, Naval Postgraduate School, June 1993.
6. Ford, Traci and Carvalho, Ron, "Parallel Implementation of an Orbital Prediction System", *EC4820 Project*, Naval Postgraduate School, 1993.
7. Geist, G.A., "Programming for Heterogeneous Networks", *Proceedings of ICASE/LaRC Short Course on Parallel Computation*, NASA, July 1993.
8. Geist, G.A. and Sunderman, V.S., "The Evolution of the PVM Concurrent Computing System", *Proceedings - 26th IEEE Compcon Symposium*, San Francisco, California, February 1993.
9. Geist, G.A. , E-Mail from Geist@msr.epm.ornl.gov to the author, 24 August 1993.
10. Hilton, C.G. and Kuhlman, J.R., "Mathematical Models for the Space Defense Center", Philco-Ford Publication No. U-3871, pp.17-28, November 1966.
11. Hoots, F.R. and Roehrich, R.L., "Space Track Report No. 3 Models For Propagation of Norad Element Sets", Aerospace Defense Command, December 1980.

12. Kozai, Y., "The Motion of a Close Earth Satellite", *Astronomical Journal*, v.64, pp.367-377, November 1959.

13. Lane, M.H. and Cranford, K.H., "An Improved Analytical Drag Theory for the Artificial Satellite Problem", AIAA Paper Number 69-925, August 1969.

14. Ostrom, Sara, "Parallelization of the Air Force Space Command (AFSPACECOM) Satellite Motion Models", *Masters Thesis*, Naval Postgraduate School, Monterey, California, March 1993.

15. Phipps, W.E., "Parallelization of the Navy Space Surveillance Center (NAVSPASUR) Satellite Motion Model", *Masters Thesis*, Naval Postgraduate School, June 1992.

16. Phipps, W.E., Neta, B. and Danielson, D.A., "Parallelization of the Naval Space Surveillance Satellite Motion Model", *Journal Astronomical Sciences*, accepted for publication, 1993.

17. PVM 3.0 Users Guide and Reference Manual, Oak Ridge National Laboratory, Oak Ridge, Tennessee, February, 1993.

18. Roy, A.E., *Orbital Motion*, p.98, 10P Publishing Ltd, 1988.

19. Sunderman, Geist, Mancheck, "The Pvm Concurrent Computing System: Evolution, Experiences, and Trends," *Parallel Computing* , submitted May 1993.

INITIAL DISTRIBUTION LIST

	No. of Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2. Library, Code 52 Naval Postgraduate School Monterey, California 93943-5000	2
3. Chairman, Department of Mathematics Naval Postgraduate School Monterey, California 93943	1
4. Commander U.S. Naval Space Command Dahlgren, VA 22448-5170	1
5. Professor B. Neta, Code MA/Nd Department of Mathematics Naval Postgraduate School Monterey, California 93943	2
6. Professor D. Danielson, Code MA/Dd Department of Mathematics Naval Postgraduate School Monterey, California 93943-5000	1
7. G.A. Geist Oak Ridge National Laboratory P.O. Box 2008 Oak Ridge, TN 37831-6367	1

No. of Copies

8. Denise Kaya — 1
HQ AFSPACECOM/CNY
150 Vandenberg St, Suite 4110
Peterson AFB , CO 80914-4110

9. Myron Matusiak — 1
2038 Juniper
Rapid City, SD 57702

10. LT Sara Ostrom — 1
14 Deene Court
Stafford, VA 22554

11. LT S.K. Brewer — 2
12881 Eagles Nest Court
Jacksonville, Florida 32246